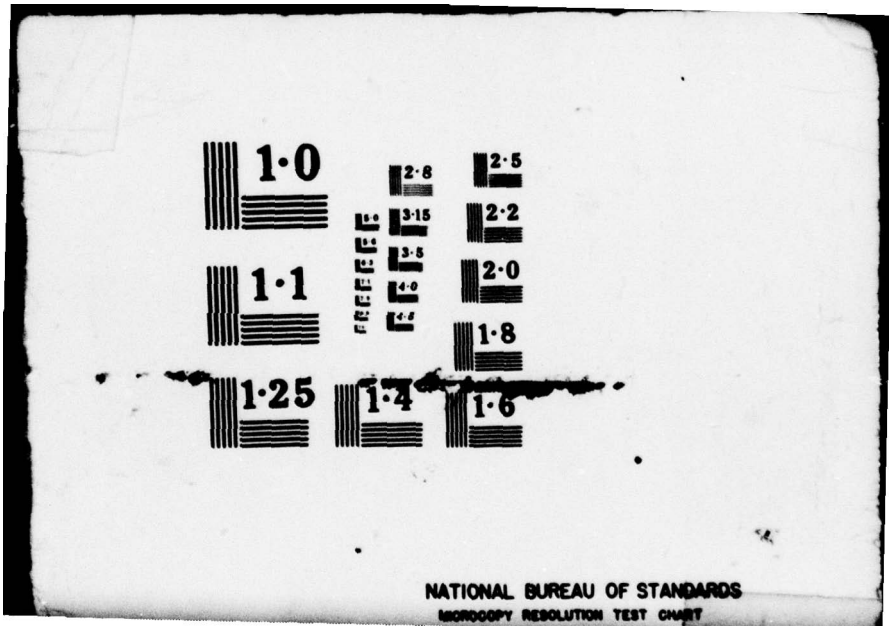
AD-A071 563 COMPUTER SCIENCES CORP FALLS CHURCH VA F/G 9/2
ATEC SOFTWARE CONVERSION STUDY.(U)
DEC 76 R BRAUNTIGAM, H REINHARDT, J STOHLER DCA100-76-C-0048
UNCLASSIFIED CSC-R4968-1-1 SBIE -AD-E100 236 NL

1 OF 2
AD
A071563


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
5·0
6·3
7·1
8·0
9·0
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

LEVEL II

ATEC SOFTWARE CONVERSION STUDY

FINAL REPORT

AD A071563

Prepared for
DEFENSE COMMUNICATIONS AGENCY
Washington, D.C.

REPORT R4968-1-1

DECEMBER 1976

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER R4968-1-1	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) ATEC Software Conversion Study		5. TYPE OF REPORT & PERIOD COVERED Final May 1976 - Dec 1976
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Ralph Brauntigam George Schaft Harry Reinhardt Robert Glicksman James Stohler		8. CONTRACT OR GRANT NUMBER(s) DCA 100-76-C-0048
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Sciences Corporation 6565 Arlington Blvd. Falls Church, VA, 22046		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS R310-76-1
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Communications Agency Washington, D.C. 20305		12. REPORT DATE December 1976
		13. NUMBER OF PAGES 160
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Defense Communications Engineering Center 1860 Wiehle Avenue Reston, VA 22091		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Automated Technical Control (ATEC) Program, Performance Assessment and Monitoring, System Control, Technical Control

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report contains the results of a study to determine which Programmable Automated Technical Control Terminal Element (PATE) software modules could be converted to a microprocessor implementation. The report defines the conversion criteria and recommends which software modules could be converted to a microprocessor implementation. The conversion feasibility is demonstrated by presenting a microprocessor configuration where conversion is recommended. Microprocessor configurations were developed for modules of the In-Service

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED 405 717

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Block #20 (Cont'd)

Quality Control Subsystem (IQCS), Out-of-Service Quality Control Subsystem (OQCS), Digital Distortion Monitoring Subsystem (DDMS) and the Modem Signal Monitor Subsystem (MSMS) functions. Cost estimates and a schedule to implement the software recommended for conversion to a microprocessor are provided. The results of a sample conversion and demonstration of the DDMS software routines implemented on a Motorola M6800 microprocessor cross-assembler and simulator are included. Microprocessor maintenance methods applicable to a microprocessor implementation of PATE software routines are discussed.

A

ATEC SOFTWARE CONVERSION STUDY

FINAL REPORT

Prepared for
DEFENSE COMMUNICATIONS AGENCY
Washington, D.C.

REPORT R4968-1-1

DECEMBER 1976

COMPUTER SCIENCES CORPORATION

6565 Arlington Boulevard

Falls Church, Virginia 22046

Major Offices and Facilities Throughout the World

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

This report contains the results of a study to determine which Programmable Automated Technical Control Terminal Element (PATE) software modules could be converted to a microprocessor implementation. The report defines the conversion criteria and recommends which software modules could be converted to a microprocessor implementation. The conversion feasibility is demonstrated by presenting a microprocessor configuration where conversion is recommended. Microprocessor configurations were developed for modules of the In-Service Quality Control Subsystem (IQCS), Out-of-Service Quality Control Subsystem (OQCS), Digital Distortion Monitoring Subsystem (DDMS) and the Modem Signal Monitor Subsystem (MSMS) functions. Cost estimates and a schedule to implement the software recommended for conversion to a microprocessor are provided. The results of a sample conversion and demonstration of the DDMS software routines implemented on a Motorola M6800 microprocessor cross-assembler and simulator are included. Microprocessor maintenance methods applicable to a microprocessor implementation of PATE software routines are discussed.

SECTION 1 - INTRODUCTION

1.1 GENERAL

The Defense Communications System (DCS) is a worldwide telecommunications system managed by the Defense Communications Agency (DCA). The overseas portion of the DCS in the Pacific, Europe and Southern Hemisphere is operated by the various military departments. This portion includes many Technical Control Facilities (TCFs) which provide the configuration control, systems management, quality control, and status reporting for the worldwide DCS. The number of technical personnel and test equipment resources greatly limits the amount of manual testing that can be practically accomplished at any one TCF due to the overwhelming number of communications circuits and associated equipment. Recognizing these physical limitations, the Government contracted with Honeywell Aerospace Division to develop a prototype system to automate the measurement and quality analysis task. This Automated Technical Control (ATEC) Program was developed to allow a computer-controlled system to accurately perform these repetitive tasks. The ATEC Program is currently in the Joint Operational Test & Evaluation (JOT&E) phase. It is anticipated that JOT&E will lead to a procurement contract for a Final ATEC Production System. Computer Sciences Corporation (CSC) was tasked to conduct a study to investigate the feasibility of converting some of the Programmable ATEC Terminal Elements (PATE) software to a microcomputer implementation.

This report provides the results of the study effort. The report defines the conversion criteria and recommends microcomputer configurations that CSC considers technically feasible for several PATE software modules.

1.2 PURPOSE

The primary objective of this study was to perform an analysis of existing software specifications and documentation for the PATE and existing microprocessor specifications to determine and recommend which PATE routines could be implemented on a microcomputer. The recommendations are based on the general

transferability of the PATE software routines to microcomputer implementation within the present PATE system architecture.

1.3 SCOPE

This study determines which PATE software routines are feasible to be implemented on microcomputers. The PATE software examined is that defined in a Honeywell Aerospace Division draft copy of the "Prime Item Development Specification for Programmable ATEC Terminal Element (PATE) of Communications Performance Monitoring - Assessment System AN/GYM-12(V) (XW)", Specification No. CP75000872 Part I, Volume I and Part II, Volume I. The architecture considered was limited to that specified in the referenced document. Pertinent excerpts from Specification CP75000872 have been incorporated into this document to define the PATE functional requirements as a basis for conversion and to make this a complete stand-alone document.

1.4 ORGANIZATION

The remaining sections of this report are organized as follows:

- Section 2 defines the criteria to be used in recommending that a PATE software routine is amenable to microcomputer implementation
- Section 3 presents the results of comparing the existing PATE software to these criteria and recommends which software should be retained in its present form and which could be converted to a microcomputer implementation. The conversion feasibility is demonstrated by presenting a microcomputer configuration where conversion is recommended.
- Section 4 presents the results of a sample conversion and demonstration of software routines for the Digital Distortion and Monitoring Subsystem implemented on a Motorola M6800 cross-assembler and simulator.

- Section 5 provides an estimate of the cost to implement the code recommended in Section 3.
- Section 6 discusses maintenance methods applicable to a microcomputer implementation of the PATE software routines.
- Section 7 summarizes the conclusions of the study effort.

SECTION 2 - CONVERSION CRITERIA

2.1 BACKGROUND

The PATE software modules are written for execution on a Honeywell 316R minicomputer. Since this computer has a larger word size, more flexible instruction set, and faster cycle time than most microprocessors, some of the PATE software will not be suitable for conversion to a microprocessor. Consequently, a baseline system has been designed for this study that contains both microprocessors and a larger control computer. Figure 2-1 depicts such a system, with the microprocessors remote from the larger computer (hereafter referred to as the "higher level" system). This higher level system will perform those PATE functions determined unsuitable for implementation by microprocessors. It will perform functions that are either common to all the microprocessors (such as output formatting) or too complex for implementation on a microprocessor (such as trending). The microprocessors will communicate with the higher level computer via phone lines or, where available, high speed data lines.

This mixed system concept is most cost-effective at current sites with a requirement for multiple PATE monitoring capability or in a remote site configuration in that one higher level computer system will serve as the point of concentration for multiple low cost microprocessor systems. The single higher level system removes most of the redundancy currently found in a multiple PATE system while retaining all of the capabilities of a single system. This is enhanced by the benefits of parallel microprocessor processing to reduce scan time at large installations such as Donnersberg, Langerkopf, or Feldberg. In this way, a configuration consisting of a higher level computer servicing one or several microprocessor-based systems can be economically superior to the current minicomputer implementation of PATE. Additional economies resulting from concentrating higher level computing at centralized locations are discussed in Section 6.

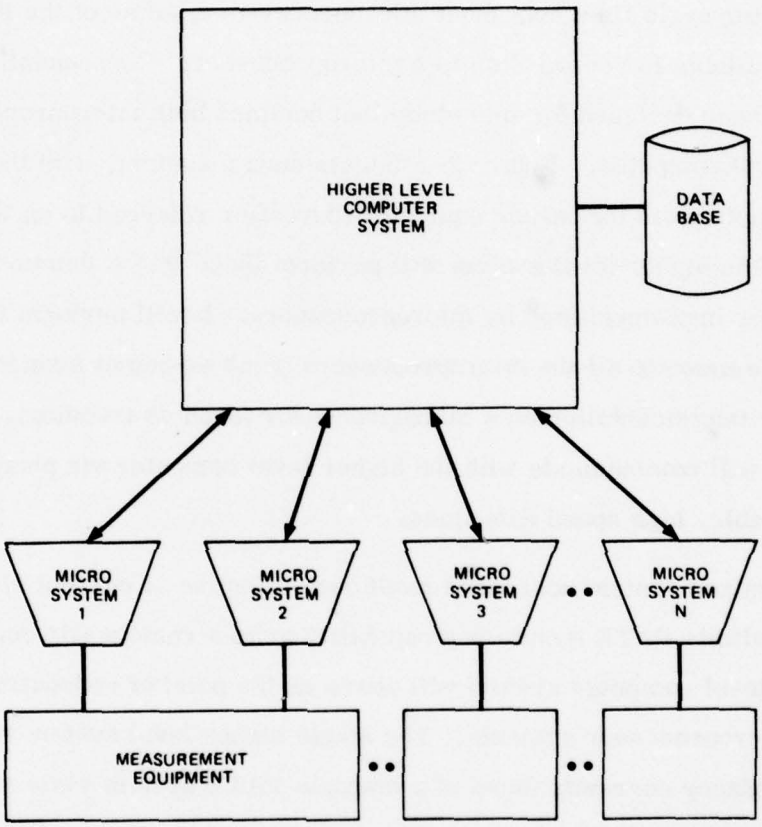


Figure 2-1. Baseline System

2.2 CONVERSION LEVEL

Four levels are possible for consideration in converting PATE minicomputer software to microprocessors. At the highest level, the entire PATE software system can be considered for conversion in bulk. At the next lower level, each major PATE function (IQCS, DDMS, MSMS, and OQCS) can be considered separately for possible conversion. Below this, each functional software module or routine (within each major PATE function) can be considered separately for conversion. At the lowest level, each computer instruction can be examined for possible conversion.

The lowest level (that of considering individual instructions for conversion) is immediately ruled out. This results from the limited communication capability between the microprocessors and the higher level computer in the baseline system configuration. Executing individual instructions within a program on different machines (parallel processing such as used in the Illiac IV computer) implies both a high degree of overhead (for control) and a large communication bandwidth (at least exceeding each machine's speed capabilities) between all of the processors. Serial data transmission, even over relatively high speed data links, falls far short of meeting this bandwidth requirement. Therefore, parallel processing at the instruction level cannot be considered where both a centralized higher level computer and remote microprocessors are involved. Consequently, consideration of software conversion (from minicomputer to microprocessor) at the machine instruction (within a program) level must be ruled out.

The highest level (that of considering the PATE software as a single entity for conversion) is also ruled out since it is too restrictive. If one minor function, algorithm, routine etc. turns out to be infeasible for conversion to microprocessors, then, working at this level, the feasibility of using microprocessors would have to be ruled out; i.e., this conversion level implies an all-microprocessor system. However, as previously discussed, a practical, cost-effective, and mixed (microprocessors and high-level machines) computer system is a possible candidate. This supports the conclusion that further consideration of this level of conversion should be ruled out. A more detailed (lower level) conversion study may, in fact, degenerate to this if all

PATE software were converted to microprocessors. However, the lower level criteria will also allow a "mixed" system, wherein some software is converted to microprocessors and other software remains at a higher level. The lower level criteria is therefore less restrictive.

Evaluation of software conversion feasibility at the major function (IQCS, DDMS, MSMS, and OQCS) level probably also suffers from the same excess of rigidity as considering the PATE software as a whole. Each of these major PATE functions somewhat parallels the others in its mix of computing requirements (i.e., handling I/O, logical and arithmetic data manipulations, data base and peripheral management, operator interaction). Thus, the situation is encountered where a whole group of varied computing requirements must be considered as a single unit for conversion to microprocessors. Since it is desirable to avoid this restriction, software conversion at this level will not be considered. Consequently, while PATE software conversion to microprocessors is discussed separately for each major PATE function, the level of conversion considered will be at the only remaining possibility, that of considering the feasibility of conversion to microprocessors at the software module level. This is done separately for each IQCS, DDMS, MSMS, and OQCS module and the results are reported by major function, for clarity.

2.3 CRITERIA FOR SOFTWARE MODULE CONVERSION

2.3.1 Attributes of Microprocessors and Larger Computers

Microprocessors (versus higher level machines) are characterized by:

1. Small Physical Size
2. Low Cost
3. Low Speed
4. Small Word Length
5. Small Instruction Sets
6. Little Software Support (in terms of operating systems, utility routines, and high level language compilers) Available.

Consequently, the following conclusions can be drawn about microprocessor-based systems. 1. They will be generally characterized by higher development costs (based on 3, 4, 5, and 6 above) but lower production and maintenance costs (based on 2) than equivalent systems employing larger computers. 2. They cannot make efficient use of high cost peripherals (particularly bulk storage devices) based on 3, 4, 5, and 6. Likewise, they are not oriented toward multitask (multiprogramming) environments; rather, they are oriented toward multiprocessor (parallel processing) systems due to 1 and 2.

Alternatively, higher level computers are characterized by:

1. Larger physical size
2. Higher hardware cost
3. Large amounts of support software available
4. Higher speed
5. Larger word length
6. Larger instruction sets.

These, in turn, result in the following attributes of higher level computers:

1. Lower development costs but higher production costs
2. Efficient use of high cost peripherals (support high speed DMA channels, efficient multilevel priority interrupt systems, etc.)
3. Oriented toward (sometimes required for cost-effectiveness) a multitask operating environment
4. Parallel processing may be employed internally to the computer, but multicomputer systems are restrictively expensive.

In the following paragraphs, these attributes are applied to PATE software requirements to establish the criteria by which to evaluate the feasibility of software conversion to microprocessors.

2.3.2 Factors Favoring Software Conversion to Microprocessors

Based on the preceding discussion of the attributes of microprocessors versus higher level computers, the following characteristics of a PATE software routine will favor its conversion to microprocessors:

1. Routine requires access to raw data - since microprocessors are small and low in hardware and maintenance costs, they can be placed physically near the source of the data, reducing communication and data acquisition problems.

2. Routine is implementable with a relatively small amount of program code - lack of support software and inefficiency in the use of bulk storage devices support this criteria.

3. Routine does not require access to a large data base - again, the inefficiency in the use of bulk storage devices dictates this criteria.

4. Routine does not require access to a large or expensive peripheral system, other than peripherals used for raw data collection - the same reasons apply here as stated in 2.

5. Routine must be processed locally at multiple sites - here, the savings in production costs offset the higher development cost characteristics of microprocessor-based systems.

6. Routine must be able to be accommodated by the computing capability of microprocessors or be logically segmentable so it can be performed by multiple microprocessors - due to the low speed, small instruction sets, and small word length characteristic of microprocessors as compared to larger processors, programs with strict timing constraints, or that require a large amount of memory may not be transferrable to a microprocessor. However, since microprocessors are inexpensive, consideration can be

given to segmenting such routines so that they may be implemented on multiple microprocessors running in parallel.

7. Routine uses peripheral which is remote from main computer - this is the result of essentially the same criteria as in 1; that is, microprocessors may be inexpensive enough to save more than their own costs by reduced communication and data handling requirements.

2.3.3 Factors Favoring Software Implementation at a Higher Level than Microprocessors

These factors are essentially the complement of those stated in Paragraph 2.3.2 and are:

1. Routine does not require access to raw data
2. Routine requires a large amount of memory - particularly when the amount is so large as to require swapping program segments between main memory and mass storage media
3. Routine requires access to a large data base
4. Routine requires support of a large peripheral system
5. Routine need not interact with remote sites or devices
6. Routine size and complexity favors larger system
7. Routine requires interaction with operating personnel who are logically located only at a centralized site.

2.4 METHODOLOGY

Now that the conversion feasibility criteria have been defined, the study procedures can be stated. Each of the PATE tasks are examined as follows:

1. Segment software into major, functionally oriented routines or modules
2. Identify attributes of each routine or module (such as memory requirements, speed requirements, data base requirements, requirements for interfacing with raw data and operating personnel, and peripheral support requirements)

3. Determine if each module's attributes favor implementation on a single microprocessor

4. If not, determine whether the module can be reasonably segmented so that each segment is suitable for implementation on a microprocessor (again, based on the criteria developed herein).

The following sections present the results of this feasibility study using the methodology described in this section.

SECTION 3 - SOFTWARE TO BE CONVERTED

3.1 INTRODUCTION

This section presents an overview of the existing PATE software and a discussion of the feasibility of division of this software between a microprocessor system and a higher level system, using the criteria given in Section 2 as the basis for the recommended software distribution. In addition, those portions of the software which are deemed feasible for implementation at the microprocessor level are presented in detail for each major PATE subsystem.

3.1.1 PATE System Overview

The present PATE software consists of operating system, application, and support programs which allow a Honeywell H316 minicomputer to function as the control element of a communications monitoring system. The software is implemented in the H316 Assembly Language (DAP-16 Mod 2) with the exception of one calculation-oriented routine which uses H316 FORTRAN IV. Table 3-1 illustrates the existing PATE software structure.

PATE software may be configured to provide one or any combination of communication system monitoring capabilities such as In-Service Voice Frequency (ISVF), In-Service Frequency Shift Key (ISFSK), In-Service Digital (ISD), and Out-of-Service Voice Frequency (OSVF).

The software is divided into tasks which perform specific functions. These tasks can be scheduled based on time, event, or by the operator. The capability to scan communications system monitor points in any order is provided. Monitor points may be grouped in ISVF, ISD, and ISFSK blocks or interleaved in any order.

When the system is started up, initialization is accomplished and the operator interaction scheduler task is scheduled. Based on the command entered, the appropriate task is scheduled to process the command. These operator interaction tasks are In-Service Quality Control Subsystem (IQCS), Digital Distortion Monitor Subsystem

Table 3-1. Existing PATE Software Family

Operating System (ADIOS)
Initialization
Scheduling
• System
• Time of Day
• Task
Scan Sequencer
Task Processor
Control
• Disk I/O
• Buffers
• ANS Interface
• Real Time Clock
Interrupt Handler
Power Failure Recovery

Support Software
System Loader
Disk Utility
Patch and Debug

Applications Software (Tasks)
Measurement
• IQCS
• DDMS
• MSMS
• OQCS
Operator Interaction
• Common
• IQCS
• DDMS
• MSMS
• OQCS
ANS Interaction
• Common
• IQCS
• DDMS
• MSMS
• OQCS
Self Test
• IQCS
• DDMS
• MSMS
• OQCS
Alarm and Trend
• IQCS
• DDMS
• MSMS
• OQCS (Alarm only)
CCSD Sort
• IQCS
• DDMS
• MSMS
Summary Report Generator
Monitor Table Executor
Schedulers
• End of Scan
• Operator Interaction

(DDMS), Modem Signal Monitor Subsystem (MSMS), Out-of-Service Quality Control Subsystem (OQCS), or common command. The first four tasks process commands which are unique to their measurement type, such as accessing and changing the appropriate data base and scheduling the measurement task. After processing a command, the operator interaction scheduler is scheduled. The common command operator interaction processes all commands that are not unique to a given measurement type. This includes accessing and changing common data base tables, scheduling time of day functions, scheduling tasks to be performed at the end of a scan, and controlling the scan sequencer. The operator interaction scheduler is also scheduled by the local teletype (TTY) control program upon receipt of an interrupt from the keyboard.

Remote control from the ATEC Nucleus Subsystem (ANS) is accomplished in much the same way as local control. The ANS interface control program reads characters from the ANS on an interrupt basis; when a complete command has been received and validated, the ANS interaction task is scheduled. If the command is common, it is processed by the ANS interaction task; if it is unique to a measurement type, the appropriate task is scheduled.

Upon command from the operator or ANS, the task processing loop is entered. Any pending time dependent tasks are scheduled at this time. If any tasks are scheduled, the highest priority is executed. If there are none scheduled, the scan sequencer is entered to determine the next monitor point to be visited and schedule the appropriate measurement task. The time of day check point then receives control.

The operator controls the scan sequencer by defining blocks of monitor points to be visited in monitor point tables and allowing the scan sequencer to process these tables. The operator can request processing of any table when desired e.g., schedule a task to be processed at a given time using the time of day scheduler.

3.1.2 Operating System

The ATEC Disk Operating System (ADIOS) is core resident, task oriented, and table driven. It consists of the following major routines: Time of Day Scheduler,

Task Processor, Scan Sequencer, Disk I/O, Local TTY Control, ANS Interface Control, Interrupt Handler, Real-Time Clock Control, Power Failure Recovery, Initialization, Output Buffers Control, and System Scheduler. To accomplish a given sequence of events, a task is scheduled, with a priority, by the user. The System Scheduler then determines which task, if any, is the next to be executed. The Task Processor loads and executes the task and returns control to the Time of Day Scheduler to determine if any time of day tasks should be scheduled. If the System Scheduler determines there are no tasks scheduled, control is passed to the Scan Sequencer. The Scan Sequencer, after scheduling the appropriate task(s), passes control to the Time of Day Scheduler.

3.1.3 Application Software

The application software is divided into the following tasks:

- IQCS Measurement - Provides measurement of voice frequency signal parameters and analysis of voice frequency performance; configures hardware to provide measurements and output results to the appropriate devices
- DDMS Measurement - Provides measurements and analysis of the digital circuit parameters; configures hardware to provide measurements; and provides output data for the required devices
- MSMS Measurement - Provides measurements of Frequency Shift Keyed and Frequency Division Multiplexed Modem Signals; configures hardware to provide measurements and output results to the proper devices
- OQCS Measurement - Performs out-of-service signal measurement and analysis; configures hardware to provide measurements and outputs data to the appropriate devices
- Common Operator Interaction - Processes commands that apply to the system level; these commands do not pertain to the specific data bases of IQCS, OQCS, MSMS or DDMS
- IQCS Operator Interaction - Provides a set of commands which allow the operator to control and maintain IQCS functions and data base

- DDMS Operator Interaction - Provides a set of commands which allow the operator to control and maintain DDMS functions and data base
- MSMS Operator Interaction - Provides a set of commands which allow the operator to control and maintain MSMS functions and data base
- OQCS Operator Interaction - Provides a set of commands which allow the operator to control and maintain OQCS functions and data base
- ANS Interaction - Decodes ANS central processor commands, executes the service routines or schedules the appropriate tasks
- Summary Report Generator - Generates end of scan, end of shift, and end of day reports
- Monitor Table Executor - Provides for processing of a Monitor Point table by the scan sequencer at a specified time
- End of Scan Scheduler - Provides task scheduling to be executed at the end of a Monitor Table scan; this function is scheduled by the Scan Sequencer when the processing of each Monitor Table is completed
- IQCS Self Test - Tests IQCS signal processing hardware and performs a computer diagnostic test
- DDMS Self Test - Tests DDMS signal processing hardware and performs a computer diagnostic test
- MSMS Self Test - Tests MSMS signal processing hardware and performs a computer diagnostic test
- Punch Tape - Dumps contents of a disk file to paper tape
- ANS IQCS Interaction - Provides a set of commands which allows the ANS to control and maintain IQCS functions and data base
- ANS DDMS Interaction - Provides a set of commands which allows ANS to control and maintain DDMS functions and data base

- ANS MSMS Interaction - Provides a set of commands which allows ANS to control and maintain MSMS functions and data base
- ANS OQCS Interaction - Provides a set of commands which allows ANS to control and maintain OQCS functions and data base
- Initialization - Provides initialization in addition to that performed by ADIOS
- MSMS Command Communication Service Designator (CCSD) Sort - Provides processing to alphanumerically sort MSMS subchannel CCSDs
- IQCS CCSD Sort - Provides processing to alphanumerically sort the IQCS monitor points CCSDs
- DDMS CCSD Sort - Provides processing to alphanumerically sort DDMS monitor points CCSDs
- Alarm Verification - Provides scheduling of appropriate measurement task when delayed alarm verification is required
- Output Monitor Table Name - Outputs monitor table name at beginning of processing of each monitor point table
- IQCS Diurnal Trend - Performs diurnal trend function for IQCS monitor points
- DDMS Diurnal Trend - Performs diurnal trend function for DDMS monitor points
- MSMS Diurnal Trend - Performs diurnal trend function for MSMS subchannels
- Operator Interaction Scheduler - Determines and schedules operator interaction task required to process a command.

The software for each task is segmented into one or more disk resident files known as subtasks. Each subtask required during task execution, along with the required data base files, are defined in configuration tables associated with each task.

3.1.4 Support Software

Existing PATE support software consists of a System Loader used to transfer subtask and data base files from disk to core, a Patch Program used for software development and debug, and a Disk Utility Program used for maintaining the PATE disk files.

3.2 RECOMMENDED SOFTWARE DIVISION

The following paragraphs discuss the division of the present PATE software into two broad general classes: higher level system software and microprocessor site software. The recommended software division is based on the results of an analysis of the PATE software which used the selection criteria delineated in Section 2 as a primary means for determining the placement of the software.

3.2.1 Higher Level Software

Present PATE software which should be implemented at a higher level is discussed in the following paragraphs.

3.2.1.1 Operating System (ADIOS) Software

In the present PATE system, as in many other computer systems, the one software package which is most dependent on the architecture of the host computer is the resident operating system. Much of its software is based on a thorough knowledge of the host computer's hardware design; therefore, any conversion of an operating system from one machine to another usually involves a complete redesign of large segments of existing code rather than a simple conversion.

Assuming that any microprocessor implementation of PATE functions will use distributed processing, the control portion of the operating system will probably assume a more prominent role. Tasks which are now done serially will often become parallel. This puts an increased demand on operating system software in the form of interrupt handling, I/O scheduling, error recovery, and other system-related functions such as data base management and dictates that future operating system

software would probably be relegated to a higher level due to the need for access to a large system of files containing both program load modules and data base information.

An additional consideration in any distributed system is the need for a central control of the communications load. In any future implementation of the PATE, the interprocessor communications load may increase if a distributed system is used. The ability of a central control to process these communications then becomes critical. If the design is such that messages must be queued when necessary, there must be some form of auxiliary storage, along with the appropriate software, available at the operating system's host computer. Present microprocessors do not justify the sophisticated software or high speed peripherals necessary for such applications.

One of the prime objectives of any new configuration using microprocessors is the reduction in the number and skill level of operating personnel at the lower levels of the system. This dictates that most, if not all, man-machine interfaces will take place at the higher level and will require considerable interaction with data base elements and communication lines. Although the system is distributed, the human interaction with it is centralized to achieve reduced manpower costs. Human interaction is traditionally a function of the operating system. It becomes increasingly important in a distributed system that the overhead associated with human interaction be minimized. This is best accomplished by maintaining control functions at a higher level where processing speed will be much faster than at the microprocessor level.

Another consideration for maintaining the ADIOS functions at a higher level is the need to provide for system expansion. The operating system provides a convenient point for both vertical and horizontal interfaces and can be segregated from the applications functions in a reasonable and logical manner.

In addition to the ADIOS functions, the present PATE software contains various tasks which are normally associated with operating systems software. Although they are not explicitly included as a part of ADIOS in the current system, they are so closely related that it is logical that they be included as part of the operating system in future system software. These tasks are:

- Monitor Table Executor
- End of Scan Scheduler
- Operator Interaction Scheduler.

Although these tasks are not considered as part of ADIOS in the present PATE software, they all heavily depend on operator-initiated actions and are part of the overall PATE control structure rather than being applications oriented. They do not require access to measured data from the Signal Processing Converter (SPC) and do not enter in any of the computations needed to provide signal parameters for alarm and trend functions. They require access to substantial portions of the data base at regular intervals for control purposes, are relevant to all of the present PATE application software, and are somewhat dependent on the operating system design.

3.2.1.2 ANS Interaction Software

The existing PATE software package contains five tasks which provide interfaces for ANS control of the PATE. Four of these tasks are associated with particular PATE measurements, i.e., IQCS, OQCS, MSMS, or DDMS measurements. The fifth processes general system commands. These tasks allow the PATE to function under control of a higher level of supervision (ANS) rather than at the local (operator) level.

In any distributed microprocessor implementation of PATE functions, the general rule will be to maintain simplicity at the microprocessor level, e.g., the microprocessor will not care whether its input command is issued from the ANS or locally; it performs functions and transfers any output data as directed. Assuming that the data base will be maintained at a higher level, the ANS Interaction software should also reside there. Much of the ANS Interaction software is concerned with the control of the communications line interface between the ANS and the PATE, decoding the received command from the ANS, and performing the necessary data base operations required for a measurement task or updating the data base. Since

the ANS Interaction tasks are heavily dependent upon both communications interfacing and data base operations, it is logical that these tasks be assigned to the higher level.

3.2.1.3 Operator Interaction Software

There are five tasks in the existing PATE software which control the interface between the local operator and the PATE. These tasks are similar to those described in Paragraph 3.2.1.2 in that they provide commands to the PATE to perform data base operations or measurement functions.

Since these tasks require the same data base interactions and logic similar to the ANS Interface software, they should also be implemented at a higher level. In addition, the concept of centralizing the man-machine interface is adhered to by a higher level implementation of this software.

3.2.1.4 CCSD Sort Software

There are three CCSD Sort tasks, one for each of the in-service measurement tasks. Each orders the CCSDs associated with its measurement task into an ascending alphanumeric sequence and displays this list on the TTY printer.

Due to their high usage of data base elements (CCSDs from monitor points) and the heavy I/O processing required, these tasks should be implemented as higher level software. They do not require access to measured data or the results of traffic parameter computations. They do have a significant auxiliary storage requirement for sorting purposes.

3.2.1.5 Alarm Verification Software

This task schedules an appropriate measurement task for delayed alarm verification. Since this is more of a control rather than applications task, it is appropriate that it be grouped at a higher level with other scheduling tasks which are closely related to the operating system.

3.2.1.6 Output Monitor Table Name Software

This task outputs the monitor table name for each monitor table point processed during a measurement. It displays the output on the TTY printer and should be located at a higher level since it provides direct operator information.

3.2.1.7 Diurnal Trend Software

There are three tasks which perform trending, one each for the IQCS, MSMS, and DDMS measurements. These tasks should be implemented at a higher level due to the complexity of the trending computations and the amount of data base interaction required. They do not require direct access to the measured data, but rather use the computed signal parameters as input.

3.2.1.8 Self-Test Software

Self Test tasks exist for the IQCS, MSMS, and DDMS. These tasks test the signal processing hardware by performing measurements and comparing them to standard parameter values stored in the data base. In addition, each task performs a memory diagnostic test of the host computer.

Since these tasks are essentially a special case of the normal measurement tasks, they can be easily initiated via commands from a higher level accompanied by the pertinent data. The computer memory diagnostic portion of these tasks is not applicable to a microprocessor implementation. The discussion of maintenance techniques in Section 6 addresses this subject in detail.

3.2.2 Microprocessor Level Software

In general, the existing PATE software which meets the criteria for implementation at the microprocessor level is contained in the four measurement tasks. These tasks contain interfaces with the various components which constitute the signal processing hardware. In addition to providing these interfaces, the measurement tasks also compute various signal parameters using the measured raw data as the initial input. The software mix contained in these tasks provides an ideal load for

a distributed microprocessor system. Certain functions performed by the measurement tasks are reasonably well-defined, easily segmentable, and are not dependent on access to large portions of the data base. Figure 3-1 shows a simplified functional block diagram of general measurement task processing. In converting a measurement task to a microprocessor implementation, the following assumptions are made for determining the performance standards required of the microprocessors:

1. Communications line between the microprocessor and the higher level system will require a maximum transmission rate of 1200 baud to provide adequate data transfer between levels.

2. Each microprocessor-based system will require approximately 2000 words of memory for communications line management software (buffer control, error detection and correction, start and stop bit sensing, etc.) and interpretation of commands from the higher level system.

3. Microprocessors directly involved in parameter computations will have a 16-bit word size for arithmetic operations and hardware or microprogrammable multiply/divide. This is necessary to maintain the required precision and speed of arithmetic operations.

4. Maximum memory capacity of individual microprocessors will be 32768 words (equivalent to 65536 bytes). This is the maximum directly addressable memory size currently available on most microprocessors.

5. The approximately 12000 words of memory used for task processing software in the existing PATE can easily be accommodated by the assumed maximum 32768 word memory capacity of a microprocessor.

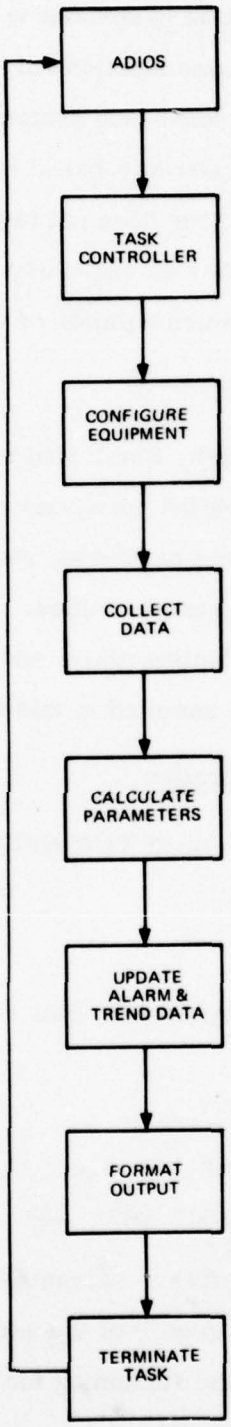


Figure 3-1. PATE Measurement Task Functional Flow

The remainder of this section is devoted to a discussion of the four PATE measurement tasks, including those functions of each task which are recommended for conversion to a microprocessor-based system. Each function considered suitable for implementation on a microprocessor-based system is discussed using a sample microprocessor configuration. This does not imply that the configurations shown are optimal designs; however, they do serve as vehicles for demonstrating the feasibility of microprocessor implementation of certain functions.

3.3 IQCS MEASUREMENT TASK

The IQCS Measurement Task, illustrated in Figure 3-2, contains the software needed for the PATE to configure the necessary signal processing hardware, sample a voice frequency signal using this hardware, and compute parameters reflecting the state of this signal based on the sample values. Additionally, it updates various IQCS related data base files reflecting alarm and trend information concerning each monitor point (channel) which is sampled in this manner.

3.3.1 Present Software Configuration

The existing IQCS Measurement Task software is divided into the following modular grouping:

- IQCS Task Control
- IQCS Equipment Configuration, Data Collection, and Parameter Calculation
- IQCS Alarm
- IQCS Trend Control
- IQCS Output Formatter
- IQCS Task Terminator.

The IQCS Task Control software serves as the overall controller of the task, including the loading from disk to core of the necessary data base files and determining the calling sequence of the remaining functions.

The IQCS Equipment Configuration, Data Collection, and Parameter Calculation software is the main processing element of the IQCS Measurement task. It configures

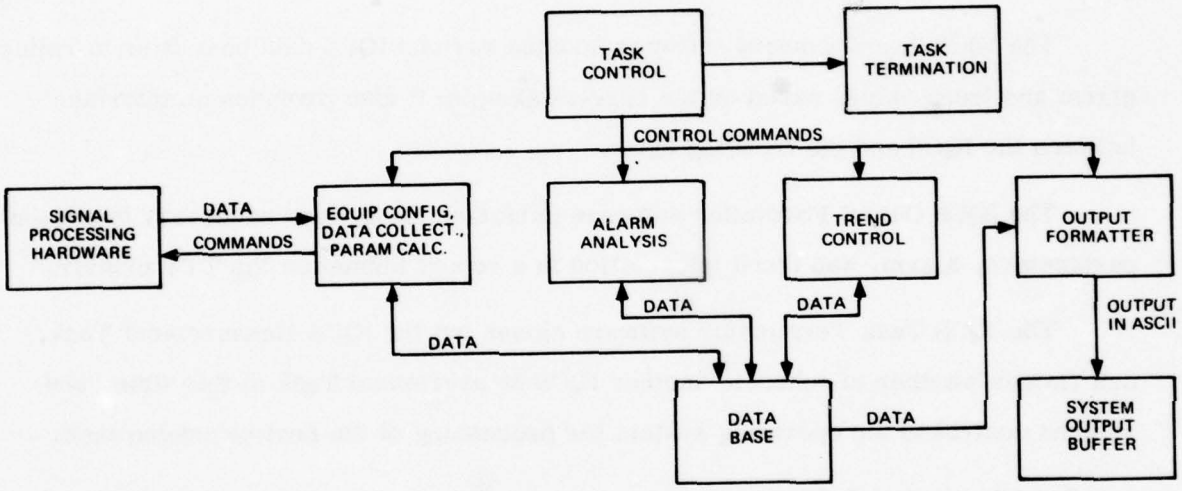


Figure 3-2. IQCS Measurement Task Block Diagram

and controls the Signal Processing hardware, collects the sampled data, computes various signal parameters, and does a spectrum analysis via a Fast Fourier Transform (FFT) algorithm.

The IQCS Alarm software determines which of the computed parameters (maximum of six) are enabled for alarm analysis and provides for immediate or delayed verification of any which are determined to be out of tolerance. This is accomplished by comparing the computed parameters to preset values maintained in data base files for alarm analysis.

The IQCS Trend Control software updates various IQCS data base files to reflect alarm and trend values based on the current sample; it also provides an interface between the IQCS and the trending tasks.

The IQCS Output Formatter software performs the actions necessary to present parameters, alarm, and trend information in a report format on the TTY printer.

The IQCS Task Terminator software closes out the IQCS Measurement Task, determines whether to schedule another IQCS Measurement Task at this time, and returns control to the operating system for processing of the next scheduled task.

3.3.2 Higher Level Software

Analysis of the IQCS Measurement Task software has shown that the following modules are best implemented at a higher level:

- IQCS Task Control
- IQCS Alarm
- IQCS Trend Control
- IQCS Output Formatter
- IQCS Task Terminator.

Higher level implementation of the IQCS Task Control software is dictated by its role as the overall task supervisor. Since it functions in this manner, it must be capable of interfacing with microprocessors at a lower level, other software modules

at the same level, and other levels to which it is subordinate, such as the ANS in the present system. In addition, it should be placed at a level which is consistent with rapid access to the data base files. This becomes especially important if future system design is oriented toward distributed processing at the microprocessor level. The concurrency of operations present in a distributed system will increase both the frequency and complexity of data base file handling operations and the need to minimize the system overhead associated with these operations. A logical and feasible method of attaining this objective is to maintain the necessary I/O routines in a central module, such as the operating system, which may also have data base management software available as an off-the-shelf item.

The IQCS Alarm software is placed at a higher level due to its need for interaction with data base files and since it follows the sampling and computational phases of the task. Also, any trend in future design toward parallel processing will increase the scope of this software. It will increase data base access demand for alarm-related items and create a need for increased interaction with the IQCS Task Control software. The alarm therefore will occur only at the higher level system.

The IQCS Trend Control software should be implemented at a higher level to maintain a logical and physical compatibility with other portions of the trending software previously assigned to this level. In addition, the amount of data base file access required by this software dictates that it be implemented at a higher level.

If it is assumed that any future system design and implementation will be oriented toward a reduction in the number of operating personnel and a more centralized man-machine interface, the IQCS Output Formatter software should be implemented at a higher level to maintain simplicity at lower levels and provide a central control of the information flow from data base files to external display devices such as TTY printers and video terminals. This software should be located at the same level as the data base management software and the operating system's common I/O handlers.

In any future system design, the IQCS Task Terminator should be located at a higher level where it would function in a control, or operating system-related capacity.

Task scheduling and operating system interface are its primary applications rather than actual participation in the IQCS Measurement Task.

3.3.3 Microprocessor Level Software

The subset of the IQCS Measurement Task software recommended for implementation on a microprocessor-based system is the IQCS Equipment Configuration, Data Collection, and Parameter Calculation module. This module controls the Signal Processing hardware interface, configures the equipment to monitor a point selected by the Scan Sequencer, collects data samples reflecting the signal state, and computes various signal parameters using the collected data. Figure 3-3 shows the existing software structure of this module.

3.3.4 Data Base Requirements

The following paragraphs contain an overview of the current IQCS Measurement Task data base requirements; a discussion of the data base items required by the IQCS Equipment Configuration, Data Collection, and Parameter Calculation module; and a description of the data base support necessary for microprocessor-based implementation of the module.

3.3.4.1 Existing IQCS Measurement Task Data Base

The subset of the current PATE data base utilized by the IQCS Measurement Task contains the following tables:

- Alarm
- Analytic Output
- Channel
- Control Load
- Parameter Combination
- Parameter Definition
- Parameter Mnemonic
- Parameter Value
- Traffic Combination
- Traffic Mnemonic
- Traffic Recognition
- Spectrum Ratio

The majority of these tables, such as those containing mnemonics, combinations, definitions, etc., contain information needed for control rather than actual

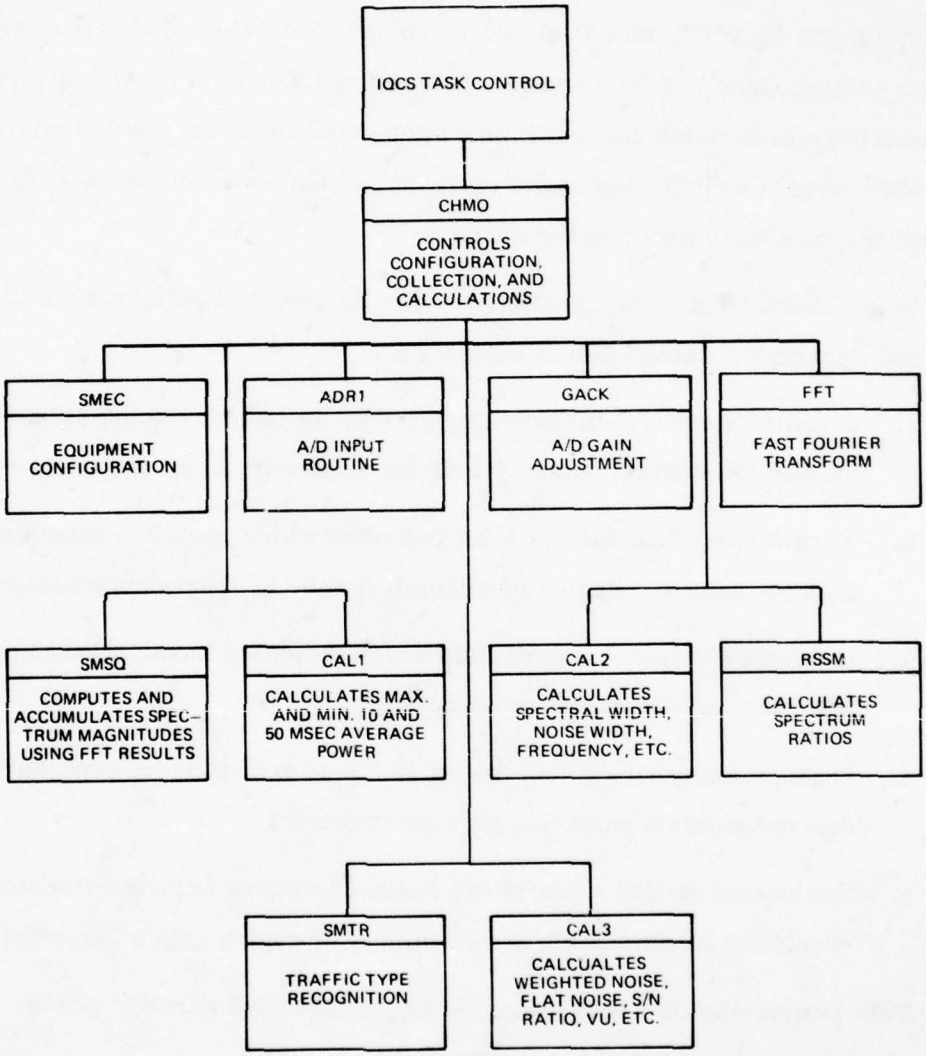


Figure 3-3. PATE IQCS Equipment Configuration, Data Collection, Calculation and Traffic Recognition Software Modules

measurement and computational-related data. In addition, the structure of these tables is based, for the most part, on the existing PATE system design; any future microprocessor-oriented system will require redesign in this area.

The bulk of the data base information needed for the IQCS Measurement Task is contained in the IQCS Channel Table file. This file contains entries for each VF monitor point (channel) in the system. Each entry has both a fixed and variable portion. The fixed portion contains general information concerning the channel characteristics along with various control items; the variable portion contains alarm and trend data for each alarmed parameter on the channel.

IQCS Channel Table data pertinent to the Equipment Configuration, Data Collection, and Parameter Calculation module are:

1. Circuit Type - a 1-bit indicator used as an index value pointing to initial system attenuation values (42 dB for transmit; 36 dB for receive)
2. Termination Selector - a 1-bit indicator which specifies termination type; high impedance bridging or special; if special, Impedance Select is used
3. Impedance Select - a 2-bit field which indicates termination impedance: 135, 600, or 900 ohms; or selectable
4. Scanner Addresses - four 16-bit fields used to connect hardware to desired monitor point (select A or C relays)
5. Test Level Points - two 16-bit fields containing adjusted transmit and receive attenuation values determined through a gain adjustment algorithm.

This information is used during the Equipment Configuration phase: the Data Collection phase has no data base requirement.

The Parameter Calculation logic requires access to the following data base files:

1. Parameter Value Table - holds computed parameter values for this monitor point

2. Traffic Recognition Table - 10 256-bit records containing recognition parameters associated with 10 traffic types

3. Tone Table - three 16-bit fields identifying traffic types which are generated tones.

This data differs from that in the Channel Table in that it is systemwide rather than being unique to each channel.

In the present PATE system, the data base files reside on disk storage and particular items are retrieved by the controlling tasks as they are required. Reference 1 contains detailed content and format descriptions of all data base files contained in the existing PATE system.

3.3.4.2 Microprocessor-Based System Data Base Requirements

In general, the data base requirements of a microprocessor-based system are similar to those of the existing PATE; however, the physical location of the data becomes critical due to the increased overhead introduced into a distributed system by the need for interprocessor information exchange. An ideal solution to this problem would be a distribution of the various data base components throughout the system, with each microprocessor having its required data base information in local memory, thereby minimizing interprocessor and interlevel communication requirements. Unfortunately, this approach is not feasible for systems with large data base requirements, such as the existing PATE. Memory requirements for a system of this type are massive (each entry in the present Channel Table file requires a maximum of 253 16-bit words). Current off-the-shelf microprocessors are usually capable of addressing a maximum of 65K bytes of memory; the technology exists for designing custom microprocessors (using bit-slice architecture) capable of addressing much larger memories, but the cost of such machines would be prohibitive when compared to current off-the-shelf minicomputers. This would destroy the concept of using microprocessors to achieve economies in future system implementations. An alternative is to include peripheral devices such as floppy discs or tape cassettes at the

microprocessor level; however, this increases the maintenance costs due to the relatively low MTBF of electromechanical versus completely electronic components and the need for trained personnel to service the peripherals. In addition, data base maintenance becomes more complex because of the decentralization of the data; the probability of human induced error becomes significant due to the increased effort required to accomplish routine data base transactions such as changes, additions, updates, etc.

As mentioned previously, the majority of the data base operations should be assigned to the higher level system. Since a large portion of the data base contains control information, and it is assumed that the higher level will exercise control of the overall system, the data base is logically placed there.

Assuming the availability of a communications line having a sufficient transmission rate (1200 baud), there should be no problem in sending data base information to the microprocessor in small segments on an "as-needed" basis. For example, the bulk of the information needed from the Channel Table file is used for Equipment Configuration. This information, consisting of 68 bits in the current PATE, could be sent to the relevant microprocessors as part of an initiate command from the higher level. The two data base files which contain systemwide information (Traffic Recognition and Tone Tables) are small enough to reside in memory at the microprocessor level (163 16-bit words in the present PATE). The Parameter Value Table file should remain at the higher level with the computed parameters being transmitted back to the higher level.

Although this study assumes that the existing PATE data base will provide the basis for future data base structure, certain areas where improvements may possibly be made which will impact overall system performance are mentioned here. For example, a large portion of the Channel Table file consists of variable length entries which contain information pertaining to parameter alarming and trending. Much of this information is needed to support the current trending algorithms. A simpler methodology for detecting degraded circuits may negate the need for holding

a large amount of trending information in the data base and reduce the Channel Table storage requirements considerably. One other improvement would be to devise a method of aggregating channels with similar characteristics rather than having data base entries for each individual channel.

3.3.5 Microprocessor Configuration

For this study, a microprocessor configuration has been developed to a level of detail necessary to demonstrate the feasibility of implementing PATE IQCS software routines on a microprocessor. This system, while not necessarily optimal, would be capable of replacing the identified PATE IQCS functions. Further tradeoff studies would be necessary, however, if an optimal design were required.

The functions which have been allocated to the microprocessor system are:

1. Control of scanner and A/D converter
2. Reading data values
3. Reducing volume of data to that suitable for transmission by appropriate calculations
4. Transmitting reduced data to the higher level system
5. Receiving commands from the higher level system
6. Exercising necessary internal controls and timing to interpret and execute commands from the higher level system.

An example of a microprocessor system to accomplish these functions is shown in Figure 3-4. A description of each of the parts of the system is given in the following paragraphs. It is necessary to assume a particular class of microprocessor to conceptualize a design. A microprocessor in the Texas Instrument TMS9900 class is assumed, since it meets the requirements defined in Paragraph 3.2.2.

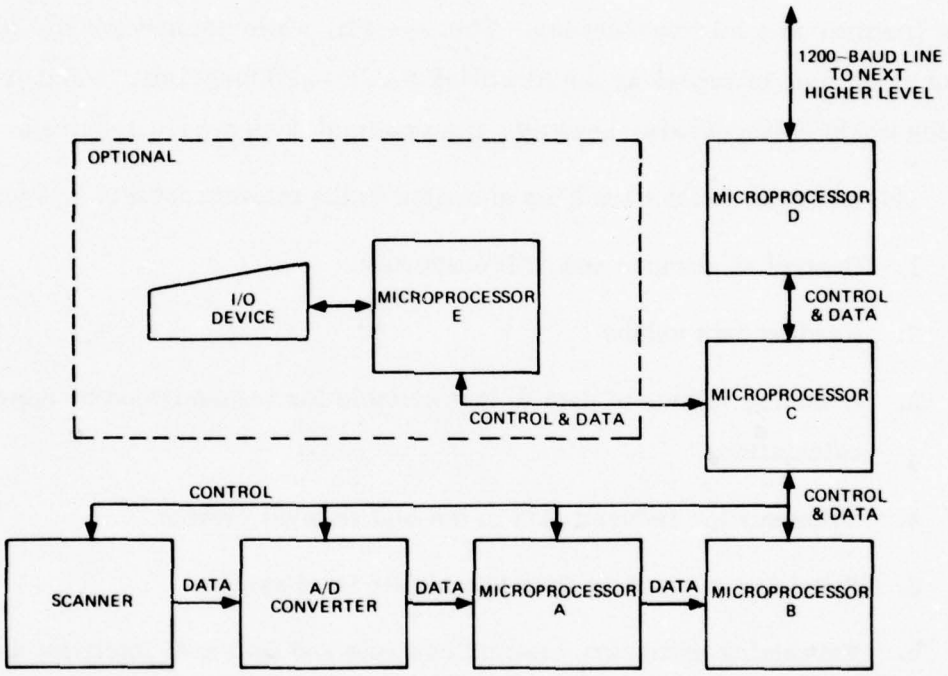


Figure 3-4. Feasible IQCS Equipment Configuration, Data Collection and Parameter Calculation Microprocessor System

3.3.5.1 Scanner and A/D Converter

For this study, the scanner and A/D converter are the same as those currently used in the PATE. The microprocessor system will interface with them in the same fashion as the H316.

3.3.5.2 Microprocessor A

Microprocessor A reads the data values from the A/D converter and does the processing required for each data value. For each set of 128 data values, microprocessor A squares, sums, averages, and detects the peak squared voltage levels. When microprocessor A completes this cycle of operations, it sends its results to microprocessor B for further processing.

3.3.5.3 Microprocessor B

Microprocessor B performs the remaining data reduction calculations. These include averaging groups of five cycles from microprocessor A, performing the FFT on every fifth cycle of microprocessor A, and further calculations on the FFT output. Because microprocessor B is an independent processing unit, it can continue its calculations while microprocessor A is going through successive cycles.

3.3.5.4 Microprocessor C

Microprocessor C is responsible for directing and coordinating the other parts of the system. When a command is received from the higher level system, it is passed to microprocessor C which decodes it and sends the appropriate control signals to the other parts of the system for execution. In the case of a normal command to scan a communications channel, microprocessor C sets up the scanner and A/D converter, starts microprocessors A and B at the appropriate times and, when the scan is complete, instructs microprocessor D to take the data from microprocessor B and send it to to the higher level system.

3.3.5.5 Microprocessor D

Microprocessor D manages the communications line. On outgoing messages, it takes data from microprocessor B or control messages from microprocessor C,

formats them, does any necessary compacting, and sends them to the higher level system. Microprocessor D performs all needed error detection and correction procedures and manages the modem.

3.3.5.6 I/O Device

The optional I/O device is envisioned as a portable keyboard terminal which is used as a diagnostic aid. Microprocessor E would contain the necessary diagnostic routines and control the I/O device.

3.3.6 Microprocessor Calculations

One of the major considerations in determining the feasibility of implementing the IQCS Equipment Configuration, Data Collection, and Parameter Calculation module on a microprocessor-based system is the ability of available microprocessors to perform the necessary computations within an acceptable time frame. That is, perform the calculations within a time span which ensures that the microprocessor-based system, in the worst case, will maintain approximately the same effective throughput as the current H316-based PATE.

The following paragraphs treat this subject in detail using an off-the-shelf microprocessor (the Texas Instruments TMS9900) as a baseline device for determining typical microprocessor instruction execution times. TMS9900 timings are derived using data available in Reference 2; H316 timings are found in Reference 3. These references also contain additional information such as architecture, instruction repertoire, etc.

The FFT algorithm analysis is taken to a finer level of detail than the other IQCS calculations due to its relatively long execution time in comparison to other calculations and its prominent role in providing parameters for follow-on calculations.

3.3.6.1 FFT Algorithm Considerations

The current FFT algorithm processes 128 points and is implemented on the H316 using approximately 266 16-bit words, including program code, constants, and storage areas. This is well within the memory capacity of most current microprocessors; they typically address a maximum of 65K bytes (equivalent to 32K 16-bit words).

The add and multiply instruction times of the TMS9900 are approximately 2.5 that of the H316. Appendix A shows this relationship. However, due to the sophisticated features of the TMS9900, such as register operations, a wider selection of addressing modes, and a comprehensive instruction repertoire, the FFT algorithm can probably be implemented using fewer instructions. The difference in speed of the H316 over the TMS9900 is estimated as closer to a factor of two for this application.

The H316 performs the FFT and level analysis in approximately 50 milliseconds; thus, it can be expected that a microprocessor in the TMS9900 class would perform these computations in 100 milliseconds in the worst case. In the present PATE, the 50-millisecond FFT and level analysis computations are interspersed with 50-millisecond data collection cycles; in the microprocessor-based system, these operations can be done in parallel rather than in series making the 100-millisecond FFT and level analysis computation time adequate in the microprocessor-based system. Consequently, implementation of the FFT algorithm on a microprocessor is feasible. In conceiving a final optimized design, it may be constructive to investigate other FFT algorithms to determine if any are more efficient than the existing algorithm.

3.3.6.2 Microprocessor System Timing

The present IQCS Data Collection and Parameter Calculation timings are illustrated in Figure 3-5 and require approximately 3.6 seconds per monitor point processed. The microprocessor system shown in Figure 3-4 should approximate this same effective throughput time to be considered a viable alternative to the present

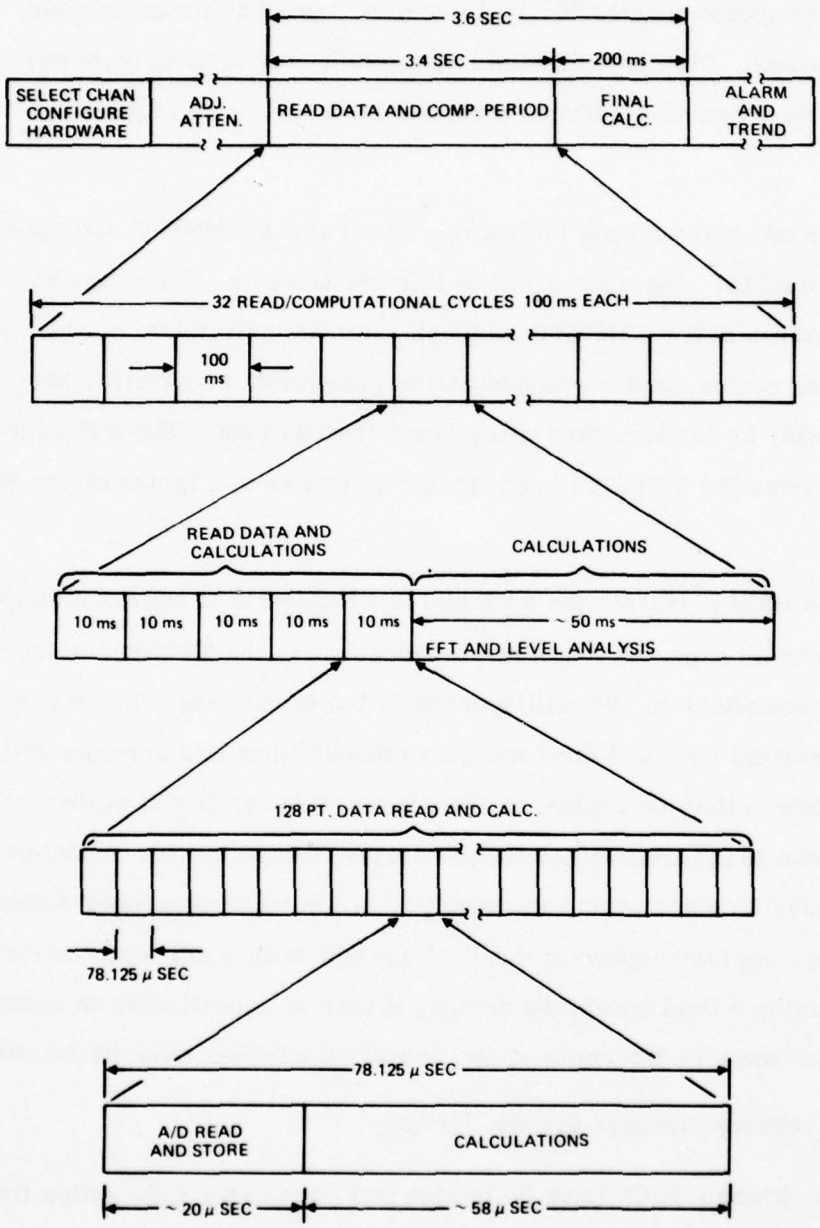


Figure 3-5. Present IQCS Data Collection and Parameter Calculation Timing

methods used to collect data and compute parameters. The present PATE performs these operations serially, interspersing data collection and computation cycles. Although it is assumed for purposes of this discussion that the microprocessors used (TMS9900 class) are slower than the H316 by a factor of two, the use of distributed processing allows an overlap of these operations, thereby maintaining approximately the same throughput time. The method used to accomplish this is described in the following paragraphs.

As shown in Figure 3-5, the present PATE performs one IQCS data collection cycle, including calculations on the raw data, in approximately 78 microseconds per data point. Assuming that the A/D sampling time shown (20 microseconds) will remain the same on the microprocessor system, approximately 120 microseconds will be required for the microprocessor to accomplish the same task. Figure 3-5, along with the microprocessor timings shown in Figures 3-6 and 3-7, indicates that the five 10-millisecond data collection periods shown for the present PATE become 20-millisecond periods in the microprocessor system. The present PATE performs the FFT and associated calculations in the 50 milliseconds following its five 10-millisecond data collection periods, thereby using 100 milliseconds for the entire process. As illustrated in Figures 3-8 and 3-9, the microprocessor time allocated for the FFT and associated calculations is 100 milliseconds. However, the FFT is performed in parallel with the next data collection cycle in the multiple microprocessor system, thereby maintaining approximately the same effective 100-millisecond throughput time as the present PATE. Figure 3-10 illustrates this technique. The total time for a complete data collection and calculation cycle on the microprocessor system, including final calculations, is approximately 3.7 seconds. This compares favorably with the time given for the present PATE to accomplish the same task.

Although this discussion is based on estimated rather than observed processing speeds, the feasibility of a microprocessor implementation of the stated task is demonstrated from a throughput time viewpoint. The overhead associated with equipment configuration and gain adjustment will require additional time, but this is insignificant when compared to the time needed to collect data and compute the parameters.

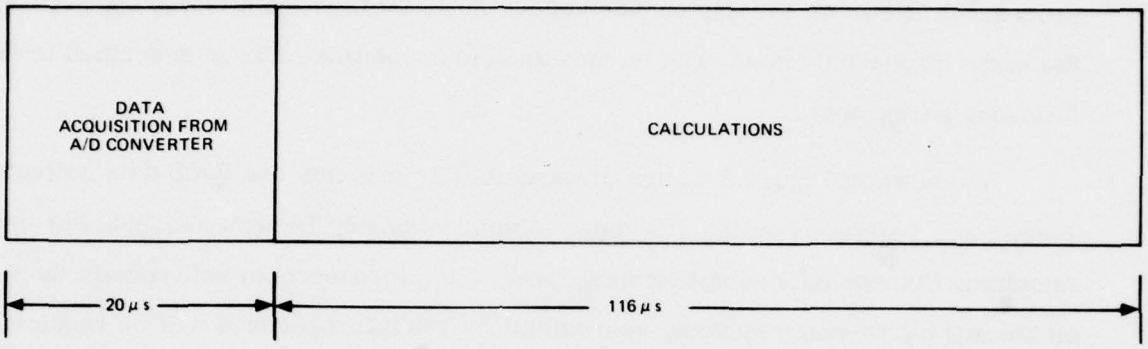


Figure 3-6. Basic Microprocessor A Cycle During Data Collection and Parameter Calculation

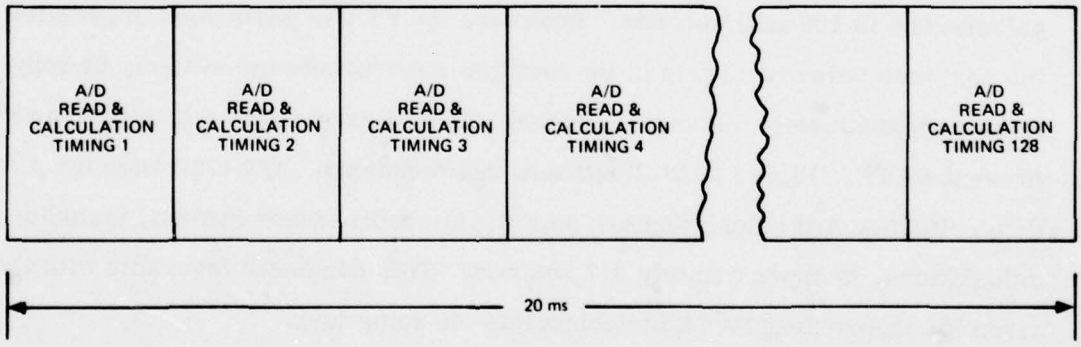


Figure 3-7. Total Microprocessor A Time for 128 Data Point Acquisition and Calculation Periods

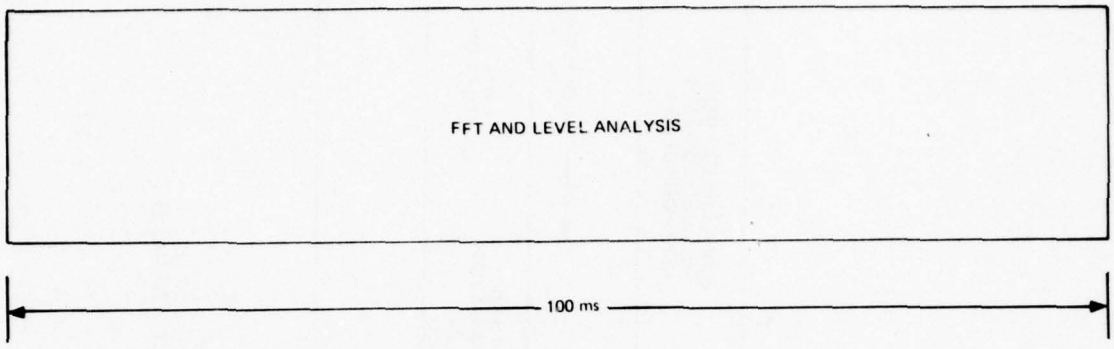


Figure 3-8. Basic Microprocessor B Cycle During FFT and Level Analysis

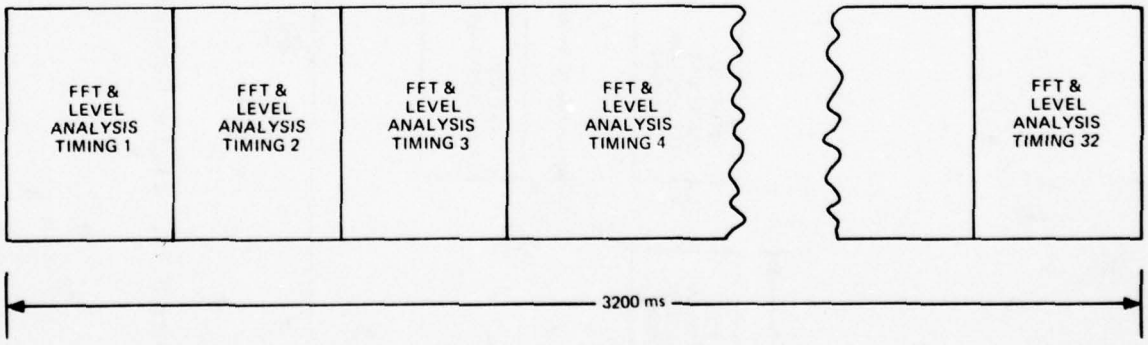


Figure 3-9. Total Microprocessor B Time for 32 FFT and Level Analysis Periods

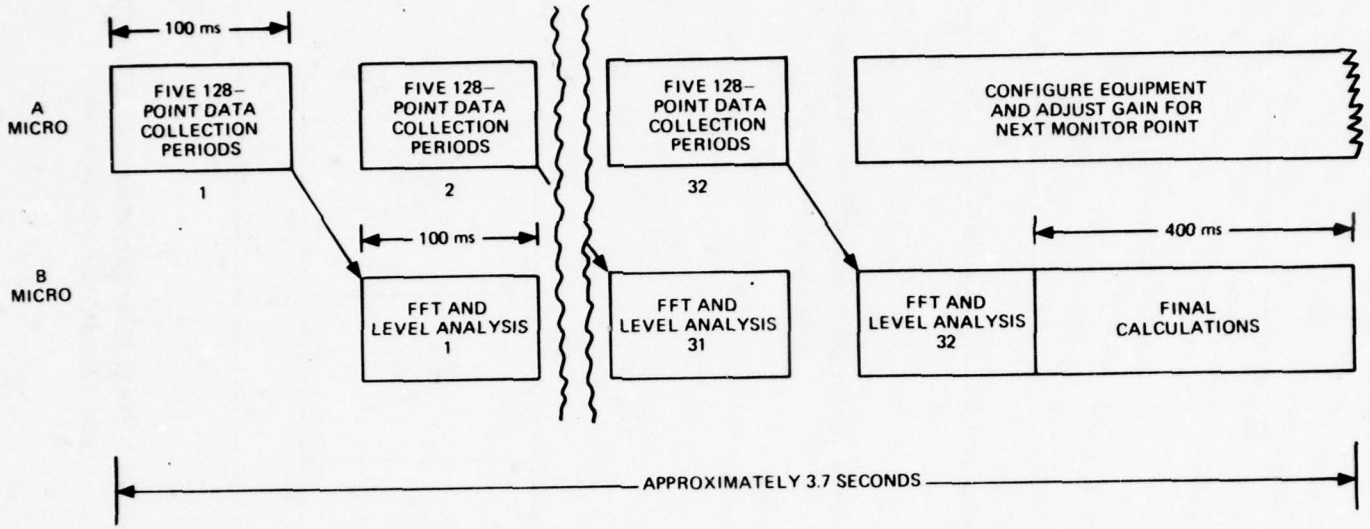


Figure 3-10. Timing Relationship Between Microprocessors A and B

3.3.7 Microprocessor/Higher Level Communications

A final consideration in determining the feasibility of implementing the IQCS Equipment Configuration, Data Collection, and Parameter Calculation module on a microprocessor-based system is the ability of the communications line to adequately support the control and data transfer message volume between the microprocessor and higher level system.

The current PATE IQCS Channel Table requires 68 bits to describe the parameters necessary to configure the signal processing hardware. This information will be contained in the data base at the higher level and will be sent to the microprocessor system over a 1200-baud line as part of a message also containing system and communication control data requiring an additional 68 bits. Consequently, the transmission time will be approximately 113 milliseconds (136 bits/1200 bps). The transmission is required once for every monitor point processing cycle (approximately every 3.7 seconds) and can be overlapped with the previous cycle.

The major load on the communication line is the transfer of the computed parameter values from the microprocessor to the higher level system. The present PATE computes the 69 parameter values shown in Table 3-2 (40 are spectrum filter values derived from the FFT output), each of which is retained in a double precision format (32 bits). If this same format is retained in the microprocessor-based system, a message containing a minimum of 2208 bits of data is required. Again, assuming a control overhead of 68 bits, the approximate transmission time over a 1200-baud line will be 1.9 seconds. This time is feasible for the microprocessor-based system which operates in parallel, especially if double buffering techniques are used. During each 3.7-second data collection and parameter calculation period, a concurrent operation could transmit the contents of the message output buffer (the parameter values computed during the previous period) while the parameter input buffer is being filled with current parameter values.

Table 3-2. IQCS Parameter Calculation Output

MNEMONIC	MEANING
SF_i	Spectrum filter value of the ith filter (i = 1 to 40)
I1	Minimum 10 ms average power
X1	Maximum 10 ms average power
X5	Maximum 50 ms average power
I5	Minimum 50 ms average power
VU	Volume units
AV	Average power
PI	Peak signal power
PA	Peak to average power
P1	Maximum 10 ms to average power ratio
P5	Maximum 50 ms to average power ratio
M1	Maximum 10 ms to minimum 10 ms power ratio
M5	Maximum 50 ms to minimum 50 ms power ratio
CN	Weighted noise, C MSG
SN	Signal to noise
NC	Noise coloration
NF	Noise frequency
HD	Harmonic distortion
SM	Spectral moment
SW	Spectral width
AF	Area factor
NW	Spectral noise bandwidth
FR	Frequency
WF	Weighted noise, 3 kHz flat
FN	Unweighted noise
VN	VU to I5 ratio
R_i	Spectrum ratio$_i$ (i = 1 to 4)

The preceding concepts show the adequacy of a 1200-baud line as a communications link between the microprocessor and higher level systems and support the feasibility of implementing the IQCS Equipment Configuration, Data Collection, and Parameter Calculation module on a microprocessor-based system.

3.3.8 *Memory Requirements*

Table 3-3 contains a brief functional description of each subroutine included in the existing IQCS Equipment Configuration, Data Collection, and Parameter Calculation module. In addition, the approximate number of lines of code is given for each subroutine. Various utilities called by the subroutines are included as an aggregated total for simplicity. The number of lines of code are derived from information contained in Reference 4. The present H316 memory requirements consist of approximately 4000 16-bit words, including utilities, and are also taken from Reference 4.

Assuming a one-to-one conversion factor when implementing the subroutines shown in Table 3-3 on a microprocessor-based system, the memory requirement, including constant and work area storage, is approximately 4000 16-bit words (8000 bytes). An additional 2000 words are assumed for communications line management software; therefore, a memory capacity of approximately 6000 words (12000 bytes) is required for implementation of the module on a microprocessor-based system.

3.4 OQCS MEASUREMENT TASK

The following paragraphs describe the OQCS Measurement Task software functions and identify which modules should remain at the higher level computer and which are recommended for microprocessor implementation. The OQCS is closely related to the IQCS in that they both deal with the same circuits, albeit under dissimilar circumstances (i.e., out-of-service versus in-service).

The OQCS Measurement Task, as illustrated in Figure 3-11, contains the software required to configure the signal sampling hardware, sample a voice frequency signal, and compute parameters to reflect the state of the sampled signal. In addition,

Table 3-3. IQCS Equipment Configuration, Data Collection, and Parameter Calculation Subroutines

NAME	FUNCTION	LINES OF CODE
ADR1	Inputs A/D data, squares, sums, and detects peak squared voltages	191
CAL1	Calculates AV, PI and maximum and minimum 10 and 50 millisecond average levels	155
CAL2	Calculates spectral width, noise width, frequency, area factor, frequency average, and parameter ratios PA, P1, P5, MI and MS	650
CAL3	Calculates weighted noise, flat noise, unweighted noise, noise frequency, and noise coloration, harmonic distortion, signal to noise ratio and VU	410
CHMO	Links channel selection, calculation, and traffic recognition routines	62
DBLO	dB calculation routine	89
FFT	Fast Fourier Transform algorithm	254
GACK	A/D gain adjusting algorithm	56
RSSM	Calculates the spectrum ratios R1, R2, R3 and R4	143
SMEC	Equipment configuration	100
SMSQ	Unfolds FFT's, computes magnitude and accumulates results	172
SMTR	Traffic recognition	151
UTILITIES	Various utility routines	1000
TOTAL		3433

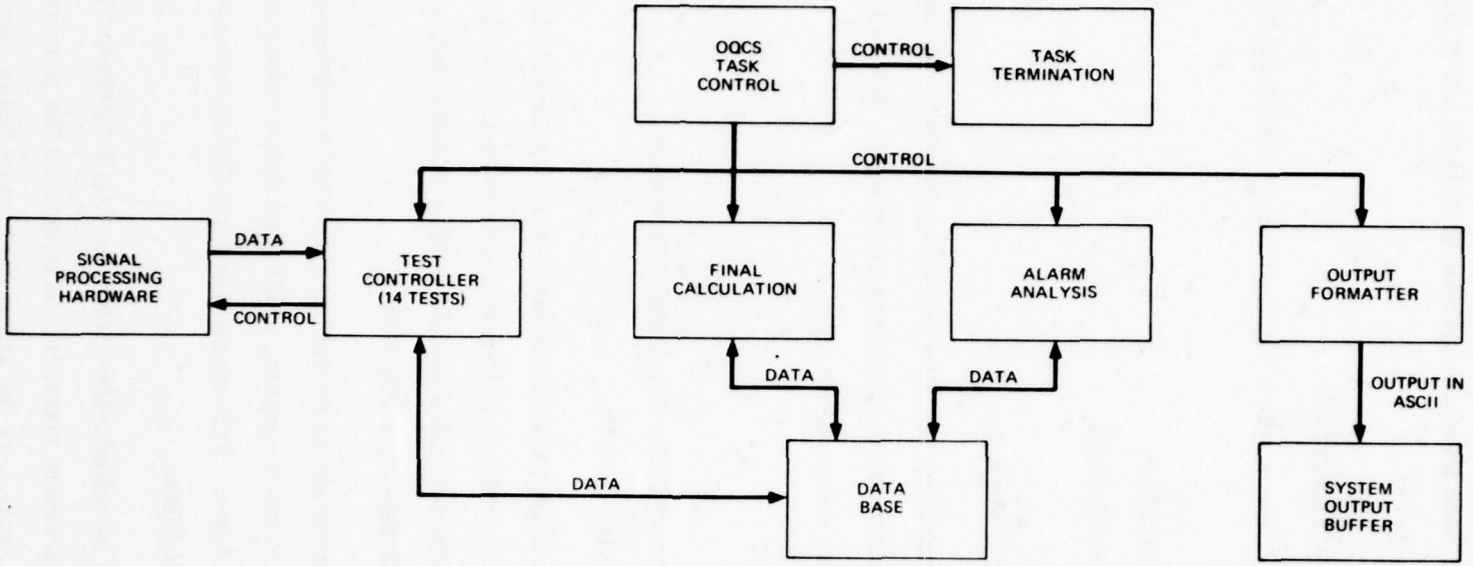


Figure 3-11. OQCS Measurement Task Block Diagram

portions of the IQCS data base files are updated to reflect alarm and trend information for each monitor point sampled.

3.4.1 Present Software Configuration

The existing OQCS Measurement Task software is divided into the following modules:

- OQCS Test Controller
- OQCS Final Calculations
- OQCS Alarms
- OQCS Output Formatter.

The OQCS Test Controller sets the print flag to OQCS, allows the output buffer to empty, obtains the control word specifying entry level, and branches to the specified entry point. The four entry points are:

1. Test Alignment
2. Self-Test
3. Test Measurement (TM) Command Processing
4. TM Impulse Noise Return.

This module is the main processing element of the OQCS Measurement Task. It is responsible for configuration, activation, and evaluation of the signal processing hardware. It specifies and collects sampled data, computes signal parameter values, and updates various IQCS data base file items.

The OQCS Final Calculations normalizes the OQCS computed values of level and envelope delay and moves frequency, level, and delay values to their respective locations within the data base. This module is logically an extension of the calculation routines used in other modules.

The OQCS Alarms is called either from the OQCS Operator Interaction Task or from one of the measurement routines. It compares the parameter values from the Final Calculations module with the alarm table thresholds selected by the operator.

It assigns a color index based on the comparison and OQCS Operator Interaction stores the color index in the ALT1 Table.

The OQCS Output Formatter performs the processing to format and output OQCS test results using the output buffer control routine common to all PATE software.

3.4.2 Higher Level Software

Aanalysis of the OQCS Measurement Task software has shown that the following modules should remain in the higher level computer:

- OQCS Final Calculations
- OQCS Alarms
- OQCS Output Formatter

Previous paragraphs have stressed the need for retaining heavy data base acitivity and operator interaction at the higher level system. The above modules interface with either the PATE operator or the ANS, or access the data base heavily, or both. The OQCS Final Calculations module relies heavily on information stored in the data base. OQCS Alarms uses the output from OQCS Final Calculations in conjunction with operator input parameters to generate its results, which are returned to the operator. OQCS Output Formatter uses PATE common output routines, the data base, and the operator console in generating its reports.

3.4.3 Microprocessor Level Software

The OQCS Test Controller software is recommended for implementation on a microprocessor-based system. With the exception of one set of routines (PTOQCS), this software is primarily concerned with hardware configuration, data acquisition, or parameter calculation. PTOQCS is used to output results from the Self-Test routine. The major routines in the OQCS Test Controller module are:

- Test Alignment
- Self-Test

- TM Command Processing
- TM Impulse Noise Return.

Test Alignment configures the hardware according to passed input parameters and inputs a test tone. Parameter values returned by the hardware are calculated against constant values from the data base and the results are output.

Self-Test configures the hardware for self-test and compares the level loss at each frequency, the envelope delay at each frequency, and parameter values with their associated limits. The self-test color is output (currently through PTOQCS).

TM Command Processing keys on a preset flag to perform one of 14 separate tests as shown in Table 3-4. Each of these tests calls a series of subroutines to configure the equipment, make the test, evaluate the results, and pass the results to OQCS Final Calculations or the operator, or both.

TM Impulse Noise Return is a special case of TM Command Processing for the Impulse Noise (IN) test sequence. It provides control to determine if sufficient time has elapsed since the beginning of the test. TM Impulse Noise Return is considered as a portion of the IN test and will not be addressed separately again.

3.4.4 Data Base Requirements

The following paragraphs contain an overview of the current OQCS Measurement Task data base requirements; a discussion of the data base items required by the OQCS Test Controller module; and a description of the data base support necessary for microprocessor-based implementation of the module.

3.4.4.1 Existing OQCS Measurement Task Data Base

Existing OQCS data base requirements are similar to those of the IQCS. The majority of the IQCS tables listed in Paragraph 3.3.4.1 are also required by the OQCS. The IQCS Traffic Recognition, Mnemonic, and Combination Tables used in conjunction with traffic type determination and alarming and trending by traffic type are not needed by the OQCS, since it functions in an out-of-service environment and

Table 3-4. OQCS Test Functions

MNEMONIC	FUNCTION
AU	Automatic
SF	Stepped Fast
SS	Stepped Slow
FM	Stepped Fast, Modulated
SM	Stepped Slow, Modulated
2T	Two Tone
1A	1 kHz (no FFT)
1B	1 kHz (FFT only)
1C	1 kHz (hit analysis)
26	2600 Hertz
DI	Dead Idle
IN	Impulse Noise
MM	Manual, Modulated
MU	Manual, Unmodulated

analyzes signals injected into the circuit by a Test Signal Source (TSS) having known characteristics.

As in the IQCS, the major data base items required are contained in the fixed portion of the IQCS Channel Table file and are used during equipment configuration and gain adjustment.

3.4.4.2 Microprocessor-Based System Data Base Requirements

These requirements are generally compatible with those stated in Paragraph 3.3.4.2 for the IQCS. The assumptions and considerations discussed in Paragraph 3.3.4.2 are also valid in an OQCS environment with the exception of the need for IQCS Traffic Recognition tables.

3.4.5 Microprocessor Configuration

To be considered for microprocessor implementation, the selected software must meet the additional criteria of low storage, simplicity, limited I/O, and acceptable execution timing. Table 3-5 presents a brief functional description of the major subroutines to be included. The storage requirements of a microprocessor-based implementation of these subroutines are discussed in Paragraph 3.4.8 and are well within the assumed maximum individual microprocessor storage capacity (32768 16-bit words).

Although the OQCS Test Controller appears to be a complex task for a microprocessor (14 different tests are required), an examination of the existing software indicates that many of the subroutines are common to more than one test and that many of the tests are merely variants of each other, e.g. the only difference between the 1A and 26 tests is the frequency of the injected test signal (1000 Hz versus 2600 Hz). Examples of this commonality are illustrated in Table 3-6. In addition, Reference 1 indicates that the existing software is very modular and recursive. As an example, Table 3-7 shows a detailed breakdown of subroutine TESTCL. Many of the subroutines called are in themselves test routines, i.e., 1ATEST, TTTEST, etc. The high degree of commonality, modularity, and recursiveness present in the OQCS

Table 3-5. OQCS Test Controller Subroutines

NAME	FUNCTION	LINES OF CODE
1ATEST	Performs 1A test; calculates phase jitter, net loss, and frequency offset at 1000 Hz	95
1BTEST	Performs 1B test; calculates harmonic distortion and signal to noise ratio at 1000 Hz	156
1CTEST	Performs 1C test; counts phase hits, amplitude hits, and dropouts at 1000 Hz	330
26TEST	Performs 26 test; calculates phase jitter, net loss, and frequency offset at 2600 Hz	entry in 1ATEST*
AUTEST	Automatic test; calculates magnitude and delay versus frequency	470
CALC1	Converts raw input data to frequency (Hz), level (dBm x 10), and delay (μs)	143
CALC2	Converts raw input data to level (dBm x 10), frequency (Hz x 10), and phase jitter (deg.)	80
CALC3	Calculates level in fixed point for greater precision	43
DELY	Converts P/D number 1 counts to microseconds	19
DITEST	Performs dead idle test; C-MSG noise, 3 kHz flat noise, noise difference, noise coloration and noise frequency are calculated	entry in 1BTEST*
DT2KH	Detects 2000 Hz and transitions to 2100 Hz	128
FREQ	Converts P/D number 2 counts to frequency in Hz	43
GIMP	Obtains scanner addresses and line impedances	35
ICTIN	Reads impulse counter	33
IGAIN	Sets initial gain	187

Table 3-5. OQCS Test Controller Subroutines (Cont'd)

NAME	FUNCTION	LINES OF CODE
INREAD	Impulse counter input	131
MNTEST	Manual test routine	284
NRMED	Normalizes envelope delay	329
NRMLV	Normalizes level	entry in NRMED*
OCHM	Initializes and calls IQCS FFT	40
OQST	Self test routine	78
RDTA	Inputs data from P/D counters for 1A and 1C tests	200
TESTCL	Test controller	532
TSST	Configures Test Signal Source (TSS)	38
TTTEST	Two tone test; calculates intermodulation distortion	110
UTILITIES	I/O handlers, interrupt routines, arithmetic format conversion	1000
TOTAL		4504

*lines of code for these subroutines are included in the count for the referenced routine of which they are an entry.

Table 3-6. OQCS Test Controller Common Subroutines

NAME	USAGE
DT2KH	Detects 2kHz and transition to 2100 Hz
AUTEST	Automatic test; sets up the TSS and sequences to produce varying test signals on a specified circuit
TTTEST	Calls 1BTEST to read data, check and/or adjust gain, perform FFTs, and calculate parameters. Also computes parameter intermodulation distortion (ID)
1ATEST/ 26TEST	1ATEST entry sets test parameters for frequency of 1kHZ, 26TEST entries sets test parameters for frequency of 2600 Hz, calls delay, RDTA, CALC2
RDTA	Reads DATA from P/D counters for 1A and 1C tests
1BTEST/ DITEST	Sets flag at entry specifying 1B (1kHz FFT) or DI (Dead Idle) test. OCHM is called to read data, adjust gain, check overflow and perform 8 FFTs via the IQCS FFT subroutine
1CTEST	Performs hit analysis
MNTEST	Measures frequency, level, and envelope delay on both modulated and unmodulated signals

Table 3-7. TESTCL Subroutine Operation

Subroutine TESTCL calls:

- TSST to configure TSS
- SETU to make measurements; SETU calls
  - TSST to output signal
  - IGAIN to set initial gain
  - DELAY to set 500 ms delay
- TTTEST to make two tone test
- DITEST to make dead idle test
- 1ATEST to make 1A test
- 1BTEST to make 1B test
- AUTEST to make AU test
- RSETC to reset scanners

Test Controller simplifies the microprocessor software development cycle and allows implementation on a microprocessor in a relatively straightforward and well-defined manner.

The amount of input/output processing required has been delineated in Paragraph 3.4.4, and is sufficiently low to generate minimal impact on the implementation.

To arrive at acceptable execution timing, and at the same time minimize costs, a possible hardware implementation is presented in Figure 3-12. As this figure shows, the OQCS implementation would take the form of one additional microprocessor (MICRO F) and associated measurement equipment attached to an existing IQCS microprocessor system. This was considered to be the best approach because of the interrelationship of IQCS and OQCS. As Paragraph 3.4.4 indicated, the OQCS uses data from the IQCS data base files. As a result, it is to be expected that an OQCS will only be used in conjunction with an IQCS.

The addition of the OQCS to the IQCS microprocessor system will involve some changes in the structure of the control microprocessor (MICRO C) and calculation microprocessors (MICRO A and MICRO B). To minimize the impact of these modifications, they will be limited to checking a flag indicating the destination of output data or input commands. Thus, microprocessor C would be modified to check for command type. If the input from the higher level computer indicated an OQCS command, the input would be passed to microprocessor F for further processing. Microprocessor F would parse the command and set up the required parameters for the scanners, A/D converters, P/D converters, and signal source equipments. A control flag would be passed to microprocessor C and from there to the scanners, A/D converter, and microprocessors A and B. The output from microprocessor B would be returned to microprocessor C for output to the higher level computer via microprocessor D.

This system will permit the OQCS to enjoy all of the benefits of parallel processing discussed for the IQCS. Additional benefits will be realized in that an IQCS can become an I/OQCS with the addition of one plug-in board and vice-versa. Further, all control, calculation, FFT processing, and scanner interfaces will not have to be redeveloped.

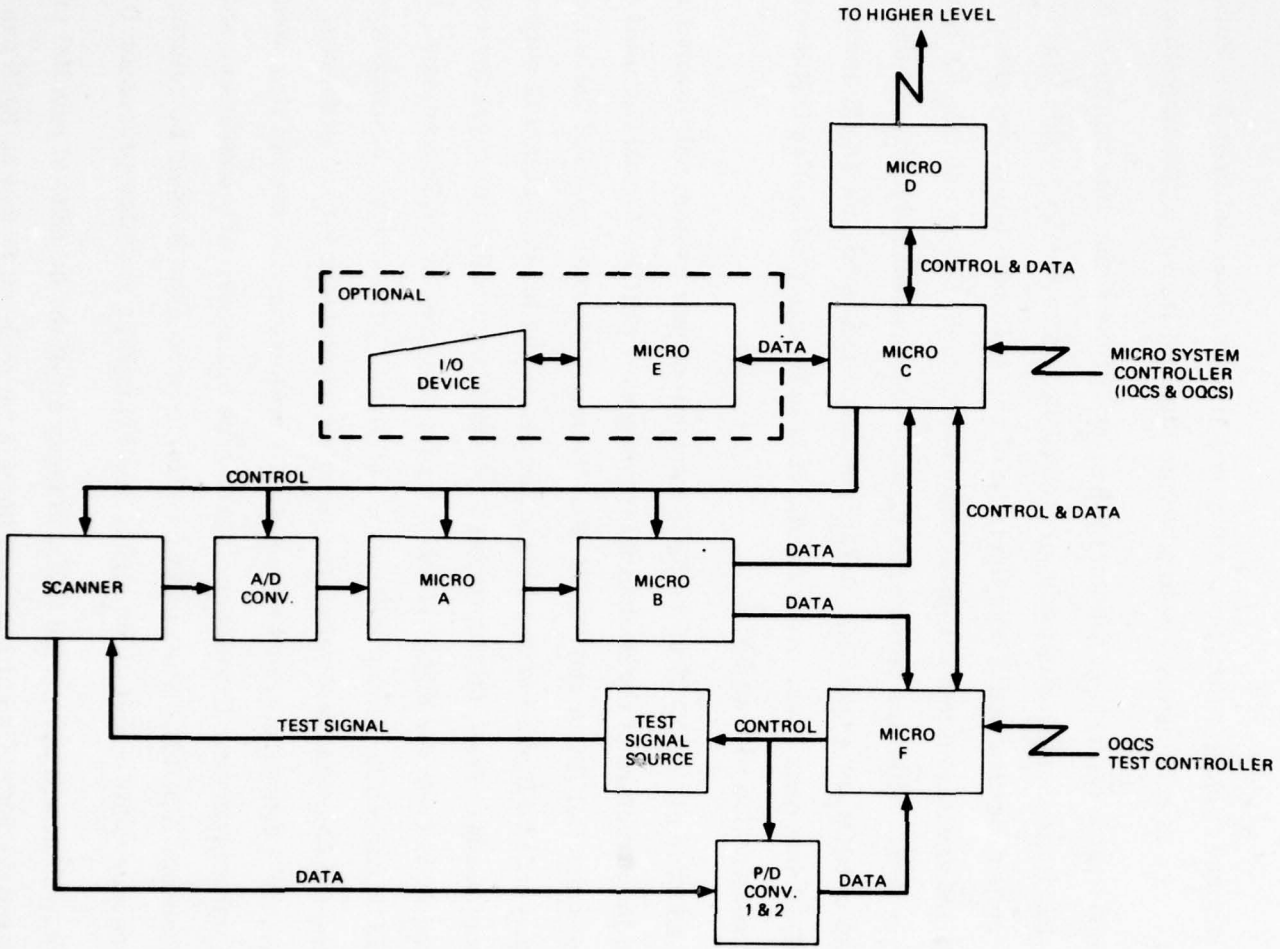


Figure 3-12. Possible I/OQCS Microprocessor Configuration

Lastly, this method of piggy-backing the OQCS onto an existing IQCS will benefit maintainability of the entire system. Loss of the OQCS will not affect the operation of the IQCS. While the converse is not true, it is felt that this is not a prohibitive restriction. The survival of the OQCS is not as inherently critical as that of the IQCS, by definition of their respective functions.

3.4.6 Microprocessor Calculations

The OQCS calculates 15 different output parameters. These parameters, along with a matrix denoting which particular TM generates them, are shown in Table 3-8. Assuming that a 16-bit microprocessor similar to the TMS9900 is used to implement the OQCS Test Controller software, it is expected that the processing speed will be slower than that of the H316 by a factor of two. However, since the tests are performed in an out-of-service environment, this does not appear to be a serious constraint, especially when it is assumed that a microprocessor-based implementation of the Test Controller will allow other portions of the OQCS Measurement task to be performed concurrently at the higher level, e.g., data base, final calculations, and alarm-related functions could be performed concurrently with microprocessor-level measurement functions. Additionally, the use of the IQCS FFT software will allow a high degree of parallelism when computing parameters which require FFTs as an intermediate step.

3.4.7 Microprocessor/Higher Level Communications

The interlevel communications load of the microprocessor-based OQCS Test Controller is essentially the same as that described in Paragraph 3.6.3 for the IQCS Equipment Configuration, Data Collection, and Parameter Calculation module except for the output parameter transmission. The IQCS calculates 40 parameters while the OQCS calculates a maximum of 15. The time savings anticipated through double buffering and parallel operations discussed in Paragraph 3.6.3 also apply here, thereby minimizing the overhead associated with communications between the two levels.

Table 3-8. OQCS Test Measurement Output Parameters

MNEMONIC	MEANING	TEST MEASUREMENT													
		AU	SF	SS	FM	SM	2T	1A	1B	1C	26	DI	IN	MM	MU
FR	Frequency	X	X	X	X	X	X	X	X	X	X			X	X
ED	Envelope Delay				X	X								X	
NL	Net Loss		X	X	X	X		X	X	X				X	X
FO	Frequency Offset	X						X			X				
CN	C-MSG Weighted Noise	X										X			
NF	Noise Frequency	X										X			
IN	Impulse Noise												X		
PJ	Phase Jitter	X						X			X				
HD	Harmonic Distortion								X						
IM	Intermodulation Distortion	X					X								
NC	Noise Coloration	X										X			
SN	Signal Plus Noise/Noise Ratio	X							X						
PH	Phase Hits									X					
AH	Amplitude Hits									X					
DO	Dropouts									X					

3.4.8 Memory Requirements

The approximate number of lines of code for each OQCS Test Controller subroutine is given in Table 3-5. Various utilities called by the subroutines are included as an aggregated total for simplicity. The number of lines of code are derived from information contained in Reference 4. The present H316 OQCS Test Controller memory requirements consist of approximately 5000 16-bit words, including utilities, and are also taken from Reference 4.

Assuming a one-to-one conversion factor when implementing the subroutines shown in Table 3-5 on a microprocessor-based system, the memory requirement including constant and work area storage, is approximately 5000 16-bit words (10000 bytes).

If this figure is added to the 6000-word estimate for the IQCS Equipment Configuration, Data Collection, and Parameter Calculation module, the combined I/OQCS microprocessor-based system shown in Figure 3-12 can be implemented using approximately 11000 words (22000 bytes) of memory.

3.5 DDMS MEASUREMENT TASK

The DDMS Measurement Task, as illustrated in Figure 3-13, analyzes the performance of a digital communications channel selected by the Scan Sequencer Task and reports its status to an operator. To accomplish this, the DDMS:

- Configures the necessary signal processing hardware to monitor the selected channel
- Samples and stores data measured on that channel
- Calculates parameters based on the measurements
- Compares the parameters to alarm thresholds to determine alarm conditions

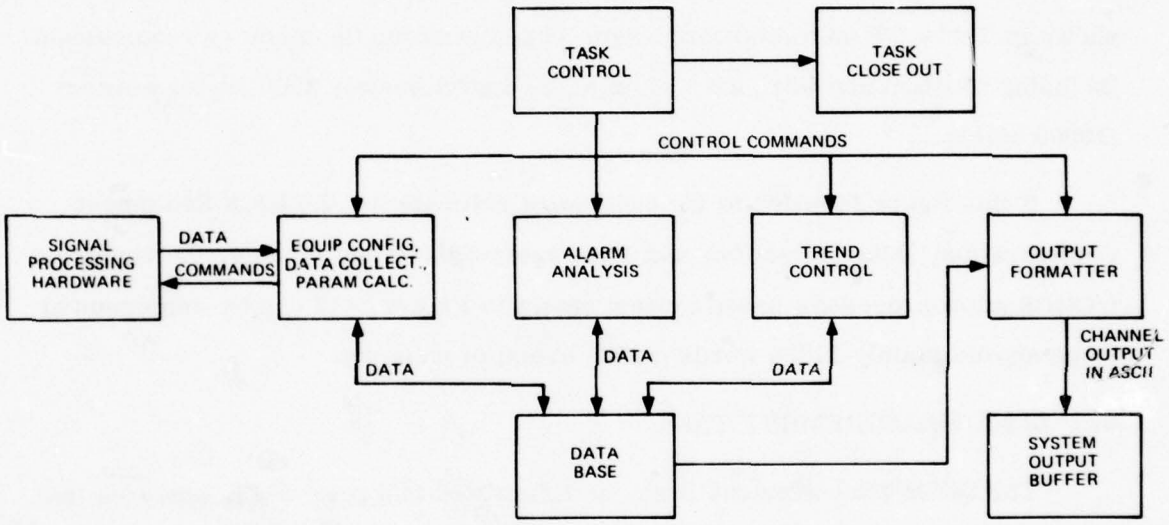


Figure 3-13. DDMS Measurement Task Block Diagram

- Computes trend indications on selected parameters
- Formats the resulting information for output.

3.5.1 Present Software Configuration

The existing DDMS Measurement Task software is divided into the following subtasks which are sequenced as illustrated in Figure 3-14.

- DDMS Task Control
- DDMS Equipment Configuration, Data Collection, and Parameter Calculations
- DDMS Alarm Analysis
- DDMS Trend Control
- DDMS Output Formatter
- DDMS Task Close Out.

The DDMS Task Control software serves as the overall controller of the DDMS Measurement Task. Its functions include loading the necessary data base files from disk to core and determining the calling sequence of the remaining subtasks.

The DDMS Equipment Configuration, Data Collection, and Parameter Calculations software controls the signal processing hardware to monitor the selected channel, reads the period to digital (P/D) counts from the period to digital converter (PDC), computes the interval counts from the P/D counts and uses them to calculate the desired signal parameters used by the Alarm Analysis, Trend Control, and Output Formatter subtasks.

The DDMS Alarm Analysis software determines the alarm state of up to six parameters for the channel being monitored. The alarm state may be either green, amber low, amber high, red low, or red high. The alarm state is determined by calculating the alarm threshold appropriate for the parameter being examined, and comparing this value with the threshold. The software also provides for immediate or delayed verification of any parameter for which an alarm state is determined.

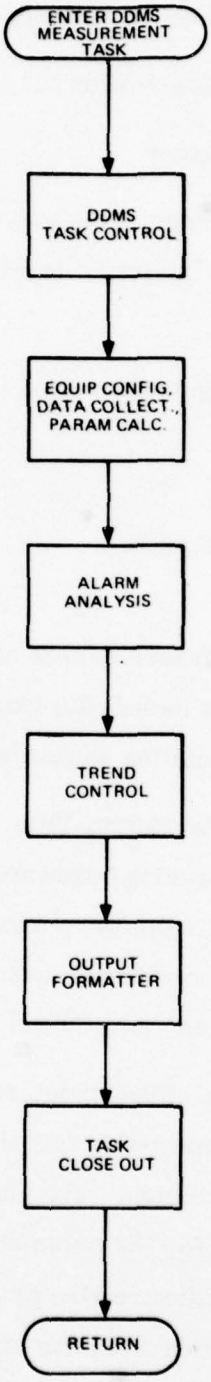


Figure 3-14. DDMS Measurement Task Flow Chart

The DDMS Trend Control provides the interface with the Trending Algorithm and updates the data base with the alarm and trend results.

The DDMS Output Formatter puts the results of the DDMS Measurement Task, including parameter, alarm, and trend information, into a report format suitable for output to a teletype.

The DDMS Task Close Out terminates the DDMS Measurement Task and determines whether to schedule another DDMS Measurement Task at this time. It then returns control to the operating system to initiate the next scheduled task.

3.5.2 Higher Level Software

Analysis of the DDMS Measurement Task software has shown that the following modules are best implemented at a higher level:

- DDMS Task Control
- DDMS Alarm Analysis
- DDMS Trend Control
- DDMS Output Formatter
- DDMS Task Close Out.

The DDMS Task Control software is not a candidate for microprocessor implementation because it controls the execution of the entire DDMS Measurement Task. This requires that it interface with all DDMS Measurement subtasks as well as the operating system. These interfaces can be done most effectively at the higher level. In addition, the DDMS Task Control loads the data base area from disk, and supplies the subtasks implemented on a microprocessor with any required data base items; thus, it should be placed at a level consistent with rapid data base access.

The DDMS Alarm Analysis software requires access to the Alarm and Trend Tables, Control Load Table, Parameter Value Table, and the fixed and variable portions of the Channel Table. It would not be feasible to put this software in a microprocessor because of the difficulty of communicating with the data base, which would be located at a higher level. The data base interaction indicates that the DDMS Alarm Analysis software should be implemented at a higher level.

The DDMS Output Formatter inputs the results of the Parameter Calculations, Alarm Analysis, and Trend Control subtasks via the data base. Its output is a formatted report on the status of the channel being monitored, which is displayed on an external device such as a teletype, line printer, or video terminal, via I/O handlers which form part of the operating system. This interaction with the data base and the operating system precludes implementation of the DDMS output formatter on a microprocessor.

The DDMS Measurement Task Close Out has task scheduling and operating system interface as its functions. It is functionally related to the operating system and therefore should be located at a higher level rather than in a microprocessor.

3.5.3 Microprocessor Level Software

The DDMS Equipment Configuration, Data Collection, and Parameter Calculation subtask is recommended for implementation on a microprocessor. The Equipment Configuration and Data Collection functions have little data base access requirement and perform few arithmetic operations. Thus, they are readily adaptable to a microprocessor. This code was converted to a microprocessor language as part of the demonstration in Section 4. Converting this part of the code to a microprocessor also eliminates the necessity of interfacing the higher level computer with the signal processing hardware. This increases system modularity and allows the higher level computer to control several units of signal processing hardware without complicating its hardware interface.

The Parameter Calculations function requires some limited data base interaction, but performs considerable computation, including floating point arithmetic. However, a 16-bit microprocessor with hardware multiply and divide, such as the TMS9900, will be capable of performing this function.

The Equipment Configuration and Data Collection functions, which require very little execution time, can be implemented on the same microprocessor. This reduces communication with the higher level computer. Rather than transmitting the entire

data buffer, which can be up to 512 words in the present PATE, it would only be necessary to transmit the resulting parameter values, which require only 20 words per channel visit.

To show that the DDMS Equipment Configuration, Data Collection, and Parameter Calculations subtask can be converted to run on a microprocessor, a possible microprocessor implementation of the subtask is presented to demonstrate that this implementation satisfies the conversion criteria defined in Section 2. Table 3-9 lists the major subroutines that would have to be converted along with the number of lines of code in each subroutine.

Figure 3-15 illustrates a possible microprocessor implementation within the baseline system environment of Section 2. The code corresponding to Equipment Configuration, Data Collection, and Parameter Calculations is implemented on a microprocessor which communicates with the DDMS Task Control and data base, both located in the higher level computer system. The microprocessor also interfaces with the signal processing hardware to transmit commands and input P/D counts.

To initiate a channel visit, the DDMS Task Control transmits necessary input arguments from the data base to the microprocessor and commands it to begin. The Equipment Configuration module sends commands to the signal processing hardware to select the appropriate channel and begin measurement on that channel. When a P/D count is available, the microprocessor receives an interrupt and control is passed to the Data Collection module, which inputs the P/D count. The Data Collection module also calculates interval counts from the P/D counts and stores them in the Data Buffer, located in the microprocessor's storage area.

The Parameter Calculations module uses the interval counts as raw data to calculate the DDMS parameters, which include:

- Baud Error
- Start Bit Error
- Bias Distortion
- Peak Distortion

Table 3-9. Lines of Code Required for DDMS Equipment Configuration, Data Collection, and Parameter Calculations

NAME	FUNCTION	LINES OF CODE
DEDEMS	Equipment Configuration, Data Collection	535
DELAY	Delay program execution for specified number of milliseconds	20
GUBOUT	Determine software timeout	50
SISP	Standard interrupt service program	110
ZAPP	Calculate DDMS parameters	117
UTILITIES	Perform arithmetic operations number conversion, manipulate floating point numbers	591
TOTAL		1423

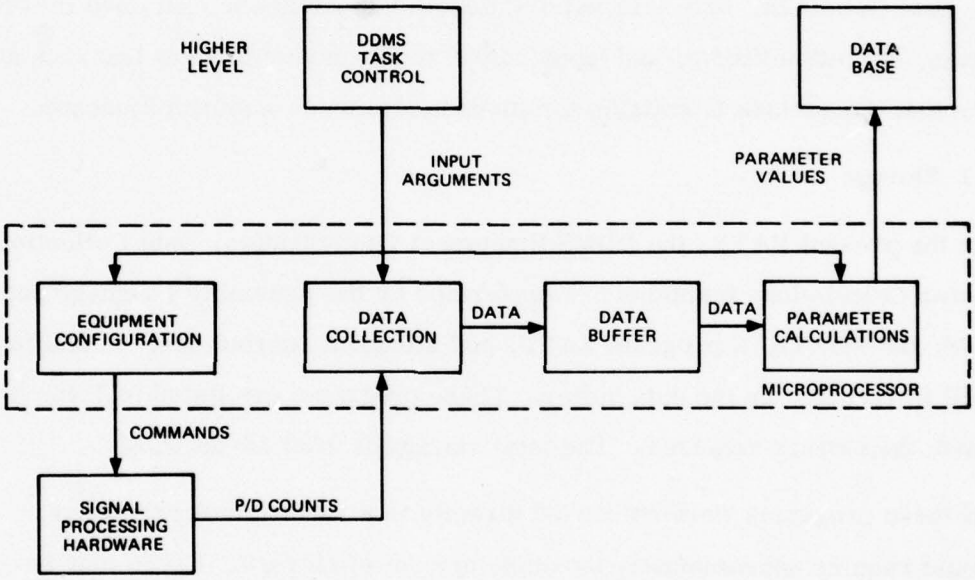


Figure 3-15. Microprocessor Implementation of DDMS Equipment Configuration, Data Collection, and Parameter Calculation Subtask

- Composite Distortion
- Fortuitous Distortion
- Average Operating Margin
- Minimum Operating Margin
- Sensitivity.

These parameter values are transmitted to the data base located in the higher level computer system upon completion of the channel visit.

The possible microprocessor implementation of the DDMS Equipment Configuration, Data Collection, and Parameter Calculations subtask is examined in terms of storage, execution timing, and input/output requirements to show that it is acceptable and that the subtask is suitable for implementation on a microprocessor.

3.5.3.1 Storage

In the present PATE, the DDMS Equipment Configuration, Data Collection, and Parameter Calculations functions are performed by the Assembly Language routine DEDEMS, the FORTRAN program ZAPP, and attendant subroutines. In addition, BUFADR is required as the data buffer. These programs are listed in Table 3-10. along with the storage required. The total storage is 2693 16-bit words.

If these programs were converted directly to a 16-bit microprocessor, they would require approximately the same amount of storage. There may be some savings in storage if ZAPP is rewritten in Assembly Language instead of in FORTRAN. Additional savings are also possible because microprocessors such as the TMS9900 have more general purpose registers and more addressing modes than the H316.

Additional storage will be required to interface with the higher level system, as discussed in Paragraph 3.2.2. The storage required for this software, plus the 2893 words required for the DDMS Equipment Configuration, Data Collection, and Parameter Calculations subtask, can be easily accommodated by a microprocessor. Thus, storage will not present a problem to the implementation of this subtask on a microprocessor.

Table 3-10. Storage Required for DDMS Equipment Configuration, Data Collection, and Parameter Calculation

Program Name	Size (16-bit words)
DEDEMS	462
ZAPP	981
DELAY	17
GUBOUT	41
SISP	83
O$22X	108
M$22X	124
A$22X	140
SQRTX	71
H$22	8
L$22	8
FLFX	79
M$11	47
BUFADR	520
TOTAL STORAGE REQUIRED	2693

3.5.3.2 Timing

There is no possibility of losing data due to the lower speed of the microprocessor in relation to the H316 because the data buffer is large enough to hold all the data received in a channel visit, and the data collection is performed by an interrupt-driven device service routine. The speed of the microprocessor may impact on the response time of the DDMS because of the longer time needed to calculate the parameters. This effect will be minimal, as demonstrated in Appendix A. Additionally:

1. Some time may be saved by writing ZAPP, the parameter calculation routine, in Assembly Language rather than FORTRAN
2. Since this subtask will occupy a dedicated microprocessor, it can be executed in parallel with other subtasks, such as alarm analysis, trending, and output, which will improve DDMS throughput
3. Depending on the interval between P/D interrupts, there may be sufficient time to perform real-time parameter calculation during data collection, even at the reduced speed.

Since the use of a microprocessor will not impact significantly on the overall speed of the DDMS, timing does not restrict microprocessor implementation of the Equipment Configuration, Data Collection, and Parameter Calculations Functions.

3.5.3.3 Input/Output and Data Base Requirements

The microprocessor will not have direct access to the data base located at the higher level system. Consequently, all its data base access will be through software located at the higher level system. Transmission of data base items will be over 1200-baud lines between the higher level system and the microprocessor.

Equipment Configuration, Data Collection, and Parameter Calculation requires 12 words of input from the data base, and produces 24 words of output, as listed in Table 3-11. Thus a total of 36 words, or 576 bits, must be transferred between the higher level system and the microprocessor for each channel visit. Assuming a

Table 3-11. Input/Output Requirements

INPUT VARIABLE	NAME	SIZE (16-BIT WORDS)
Baud, Bit Per Second	BAUD	2
Mark Orientation Indicator	MOIN	1
Code Length	CODE	1
Sample Size	SIZE	1
Number of Candidate Calculations	CANS	1
Maximum Dwell Time	DWEL	1
Channel Number	CHAN	1
Signal Type Index	TYPX	1
Slicer Setting	SLSC	1
Maximum Signal Interval	MAXI	1
Input Mode	MODE	1
TOTAL		12
OUTPUT		
Number of Space to Mark Transitions	NS2M	1
Number of Mark to Space Transitions	NM2S	1
Baud Error	BE	2
Start Bit Error	SE	2
Bias Distortion	BD	2
Fortuitous Distortion	FD	2
Composite Distortion	CD	2
Peak Distortion	PD	2
Average Operating Margin	AM	2
Minimum Operating Margin	MM	2
Completion Code	CC	1
Number of Spurious Transitions	NST	1
Mean Duration of Spurious Transitions	SBAR	2
Standard Deviation of Spurious Transitions	SDEV	2
TOTAL		24

1200-baud line connecting the systems, it will take approximately 480 milliseconds to access the data base buffer. Using a double buffering technique and an interrupt-driven communications line service routine, the input/output will not delay the microprocessor significantly. Consequently, the input/output requirement does not preclude implementing DDMS Equipment Configuration, Data Collection, and Parameter Calculation on a microprocessor.

3.6 MSMS MEASUREMENT TASK

The MSMS Measurement Task analyzes the performance of frequency shift-keyed and frequency division-multiplexed modem signals on communications channels selected by the Scan Sequencer. It configures the hardware to provide measurements on the selected channel, samples and stores data measured on that channel, calculates parameters based on the measurements, compares the parameters to alarm thresholds to determine alarm conditions, computes trend indications on selected parameters, and formats the resulting information for output.

3.6.1 Present Software Configuration

The existing MSMS Measurement Task software is divided into the following subtasks:

- MSMS Task Control
- MSMS Equipment Configuration, Data Collection, and Parameter Calculations
- MSMS Alarm
- MSMS Trend Control
- MSMS Output Formatter
- MSMS Task Close Out.

The MSMS Task Control serves as the overall controller of the MSMS Measurement Task. Its functions include loading the necessary data base files from disk to core and determining the calling sequence of the remaining subtasks.

The MSMS Equipment Configuration, Data Collection, and Parameter Calculations controls the signal parameter converter hardware and configures the equipment associated with monitoring the subchannel selected by the Scan Sequencer. It also reads data from the analog to digital (A/D) converter and two period to digital (P/D) converters, and calculates parameters based on the data read. These parameter values are used by the Alarm, Trend Control and Output Formatter subtasks.

The MSMS Alarm software determines the alarm state of up to eight parameters for each data type per subchannel. The alarm state is determined by calculating the alarm threshold appropriate for the parameter being examined, and comparing the parameter value with the threshold. The MSMS Alarm software also provides for immediate or delayed verification of any parameter for which an alarm state is determined.

The MSMS Trend Control provides the interface with the Trending Algorithm and updates the data base with alarm and trend results.

The MSMS Output Formatter puts the MSMS Measurement Task results, including parameter, alarm, and trend information, into a report format suitable for teletype output.

The MSMS Task Close Out terminates the MSMS Measurement Task and determines whether to schedule another MSMS Measurement Task at this time. It then returns control to the operating system to initiate the next scheduled task.

3.6.2 Higher Level Software

Analysis of the MSMS Measurement Task software has shown that the following modules are best implemented at a higher level:

- MSMS Task Control
- MSMS Alarm
- MSMS Trend Control
- MSMS Output Formatter
- MSMS Task Close Out

The MSMS Task Control is not a candidate for microprocessor implementation because it controls the execution of the entire MSMS Measurement Task. This requires that it interface with all the MSMS Measurement subtasks as well as the operating system. These interfaces can be done most effectively at the higher level. Also, the MSMS Task Control loads the data base area from disk, and it would supply the subtasks implemented on a microprocessor with any required data base items. This requires that it should be placed at a level consistent with rapid data base access.

The MSMS Alarm software requires access to the Alarm and Trend Tables, Control Load Table, Parameter Value Table, and the fixed and variable portions of the Channel Table. It would not be feasible to put this software in a microprocessor because of the difficulty of communicating with the data base, which would be located at a higher level. This data base interaction thus requires the MSMS Alarm software to be implemented at a higher level.

The MSMS Trend control software uses parameter values, historical alarm and trend information, alarm data, and the Channel Table to determine which parameters are to be trended. The large amount of data base access requires this software to be implemented at a higher level.

The MSMS Output Formatter inputs the results of the parameter calculations and the Alarm and Trend Control subtasks via the data base. Its output is a formatted report on the status of the channel being monitored, which is displayed on an external device such as a teletype, line printer, or video terminal, via I/O handlers which form part of the operating system. This interaction with the data base and the operating system preclude implementation of the MSMS Output Formatter on a microprocessor.

The MSMS Measurement Task Close Out has task scheduling and operating system interface as its functions. It is functionally related to the operating system and therefore should be located at a higher level rather than in a microprocessor.

3.6.3 Microprocessor Level Software

It is feasible to implement only part of the MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask on a microprocessor. This subtask can be broken down into two separate modules: MSTM subroutine, which monitors the available subchannels of a specified channel to determine if they are all present; and MSCHMO subroutine, which configures the equipment for a particular subchannel and calls other subroutines to collect data from that subchannel and calculate parameters from that data. MSCHMO can be implemented on a microprocessor and MSTM should be implemented at the higher level, as shown in the following paragraphs.

3.6.3.1 MSTM Subroutine

The MSTM subroutine fetches data from the Channel Table as well as the Subchannel Table. It utilizes subroutines FNDSCH, MGETFX, and MPUTFX to find and update the Subchannel Table for each subchannel in the channel monitored and formats and prints messages for each subchannel monitored. Because of the large amount of data base access and the interaction with the operating system involved in printing the messages, the MSTM subroutine should be implemented at the higher level.

3.6.3.2 Software to be Implemented on a Microprocessor

The portion of the MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask that can be implemented on a microprocessor is illustrated in Figure 3-16. The MSMS Task Control supplies commands and inputs to this subtask. The subtask accesses various tables in the data base, such as the Channel Table, Subchannel Table, and Control Load Table. The Equipment Configuration function is performed by the MSCHMO subroutine, which selects the scanner position, initial gain settings, impedance, and subchannel; connects the modules of the signal processing hardware together to form the desired configuration; selects the A/D conversion rate; and selects the clock for the P/D converters.

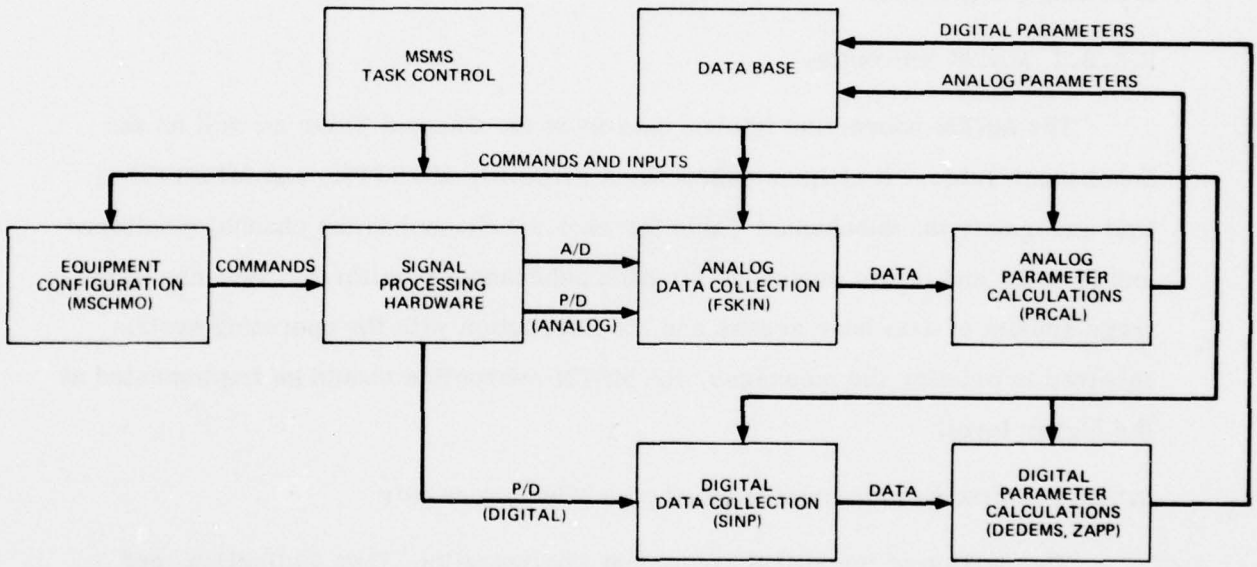


Figure 3-16. MSMS Equipment Configuration, Data Collection, and Parameter Calculations Block Diagram

The signal processing hardware produces A/D values and P/D counts which are read by subroutines FSKIN and SINP. FSKIN collects analog data from the A/D converter and the analog P/D converter; SINP collects digital data from the digital P/D converter. The required parameters are calculated by subroutine PRCAL, which calculates analog parameters from data collected by FSKIN; and subroutines DEDEMS and ZAPP, which calculate digital parameters from data collected by SINP. The parameter values are stored in the parameter value table in the data base.

3.6.3.3 Microprocessor Implementation

Figure 3-17 illustrates a possible microprocessor implementation of the MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask. Table 3-12 lists the required subroutines, along with a brief functional description and the number of lines of code in each subroutine. The data base resides at the higher level, as does the MSMS Task Control. Subroutines MSCHMO, FSKIN, PRCAL, SINP, DEDEMS, and ZAPP are implemented on one microprocessor, to reduce interprocessor communication. Because of the large amount of arithmetic involved in the parameter calculations, a 16-bit microprocessor, such as the TMS9900, will be required. All access to the data base is performed by the MSMS Task Control at the higher level system, and the Task Control sends commands and inputs to the microprocessor over a 1200-baud line. It also receives the parameter values and other outputs from the microprocessor and stores them in the data base.

The microprocessor has an interface program to communicate with the higher level system, which receives inputs, interprets commands, and transmits outputs. The other software modules have the same functions as in the present PATE.

- MSCHMO configures the signal processing hardware
- FSKIN collects A/D values and analog P/D counts
- PRCAL calculates the analog parameters
- SINP collects the digital P/D counts
- DEDEMS and ZAPP calculate the digital parameters.

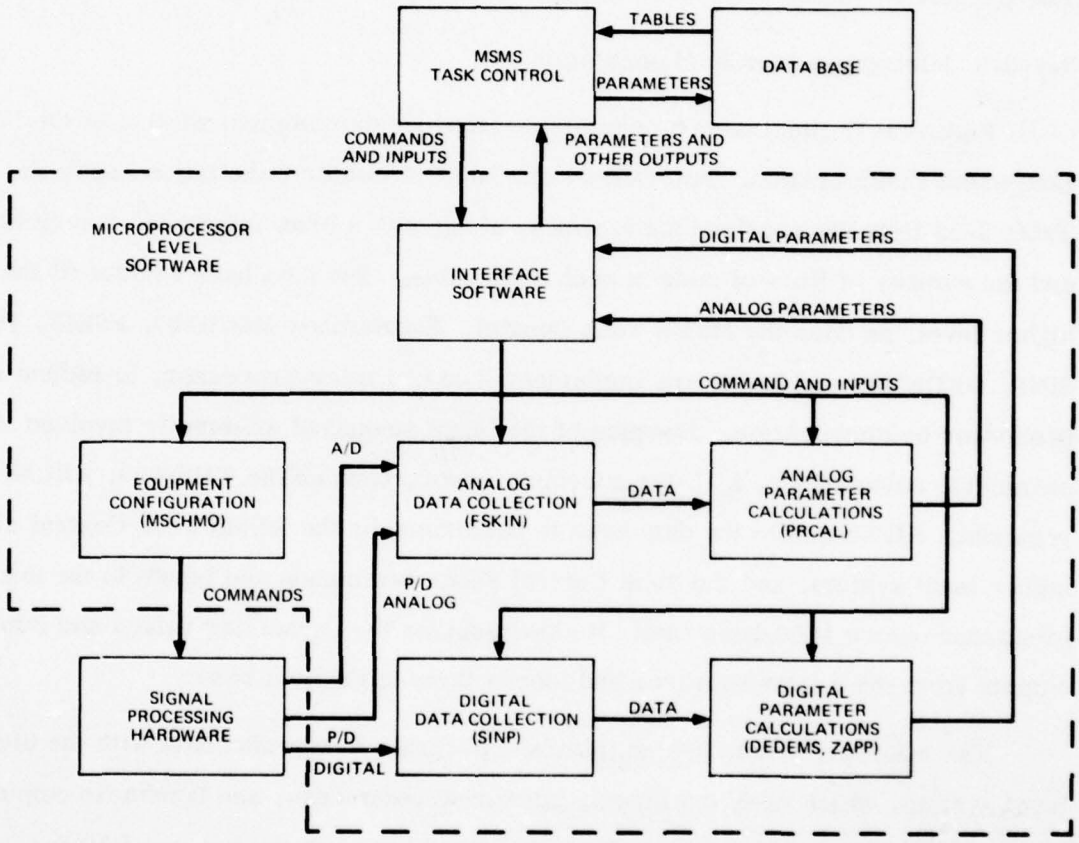


Figure 3-17. Microprocessor Implementation Data Flow for MSMS Equipment Configuration, Data Collection, and Parameter Calculations

Table 3-12. MSMS Equipment Configuration, Data Collection, and Parameter Calculations Subroutines

NAME	FUNCTION	LINES OF CODE
ADR1	Performs A/D data input	187
DEDEMS	Collects digital P/D data, calls ZAPP to calculate parameters	493
FSKIN	Inputs data from analog P/D and A/D for an MSMS subchannel	308
GACK	Reads overflow on A/D and compares with allowable number of overflows	59
GTOR	Selects gain, channel, impedance, filter, A/D rate	114
GUBINI	Timeout subroutine	50
LEVEL	Calculates level of composite channel	77
MSCHMO	Configures equipment for MSMS subchannel measurement, controls MSMS subtask	203
PRCAL	Calculates analog MSMS parameters	273
ZAPP	Calculates digital MSMS parameters	114
UTILITIES	Performs arithmetic operations, number conversion, manipulate floating point numbers	901
TOTAL		2779

To demonstrate that the microprocessor configuration of Figure 3-17 is a feasible implementation, it is necessary to show that it satisfies the storage requirements, input/output requirements, and timing requirements of the MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask.

3.5.3.2.1 Storage Requirements

The subroutines and buffers required for the MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask listed in Table 3-13 require 3910 16-bit words in the present PATE. Adding 2000 words of storage needed for the interface software yields a total of 5910 words required, which can be easily accommodated by a 16-bit microprocessor.

3.5.3.2.2 Input/Output Requirements

The MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask requires inputs from the Channel Table, Subchannel Table, Control Load Table, and Baud Selector Table, as well as several flags and other data base information. This amounts to 26 words of input as listed in Table 3-14, assuming that the microprocessor-based system uses the same format and data base organization as the present PATE. Table 3-14 also lists the 18 words of output produced by this subtask, which include the data type, level error, and eight parameter values.

The subtask thus requires 44 16-bit words, or 704 bits, of input and output per subchannel visit. Assuming a single 1200-baud bidirectional line between the microprocessor and the higher level computer system, this input/output will take approximately 600 milliseconds. By using double buffering techniques device handlers, the input for the next subchannel visit and the output from the last subchannel visit can be performed simultaneously with the current subchannel visit. Consequently, the input/output requirements will not impact significantly on the MSMS throughput and do not preclude a microprocessor implementation of the MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask.

Table 3-13. MSMS Subroutines

SUBROUTINE	SIZE (16-bit words)
MSCHMO ($SMSCM)	186
ADR1 ($SADR1)	396
DEDEMS ($SMSDC)	459
DELAY ($SDDLY)	17
FLFX (SFLFX)	79
FSKIN ($SIFSK)	310
GACK ($SGACK)	50
H$22 ($SHF22)	16
L$22 ($SLF22)	13
LEVEL ($SMSLE)	82
PRCAL ($SPRCA)	263
A$22 ($SSF2X)	140
FILL ($SFILL)	19
M$22X ($SMFLX)	124
SQRTX ($SSQRT)	71
D$22X ($SDF22)	108
DFLOAT ($SDFLT)	21
GUBINI ($SGNBO)	41
N$22 ($SNF22)	10
ZAPP ($SMZAP)	981
ALOGIO ($SALOGX)	176
DIVI ($SDIVI)	21
ARG$	15
C$12	25
F$ER	27
BUFFER	360
TOTAL	3910

Table 3-14. MSMS Inputs and Outputs

INPUTS	OUTPUTS
Monitor Point	Level Error
Current Mode	Data Type
Mark Orientation	* RMS Mark Frequency Error
Dynamic Baud Selector	* RMS Space Frequency Error
Wide/Narrow Shift	* Average Mark Frequency Error
Signal Type	* Average Space Frequency Error
Character Length Selector	* Differential Mark and Space Power
KW26 Selector	* Subchannel Relative
Number of Subchannels	* Bias Distortion
Transition Selector	* Peak Distortion
Baud Selector	
Level of which Subchannel is Defined as Missing	
Wide High Shift	
Wide Low Shift	
Narrow High Shift	
Narrow Low Shift	
Self-Test Flag	
Total Monitor Flag	
Initial Gain Setting	
Gain Set	
TSM Index to DAL Table	
Subchannel Number	
TSM Subchannel Number	
TSM Subchannel Type	
*Baud Rate	
Number of Overflows before Gain Adjustment	

* These items consist of two 16-bit words; all others are contained in one 16-bit word.

3.5.3.2.3 Timing Requirements

The H316 is 2.5 times as fast as a microprocessor such as the TMS9900, as demonstrated in Appendix A; therefore, the microprocessor implementation will require an upper limit of 2.5 times as much execution time to perform a subchannel visit as the current PATE. There are several factors which could serve to reduce this ratio:

1. Most 16-bit microprocessors have more addressing modes and more general purpose registers than the H316. Efficient use of these features can reduce execution time.

2. Many calculations performed by this subtask are interlaced with data collection. Depending on the baud rate of the data being collected, there may be sufficient time during data collection to perform required calculations at the slower execution speed of a microprocessor, without delaying the overall speed of the MSMS

3. In the H316, this subtask is performed sequentially with other MSMS subtasks, such as Alarm Analysis and Trend Control. In a microprocessor implementation, it can be performed in parallel with other subtasks because it is executed by a separate processor. This could result in significant time savings.

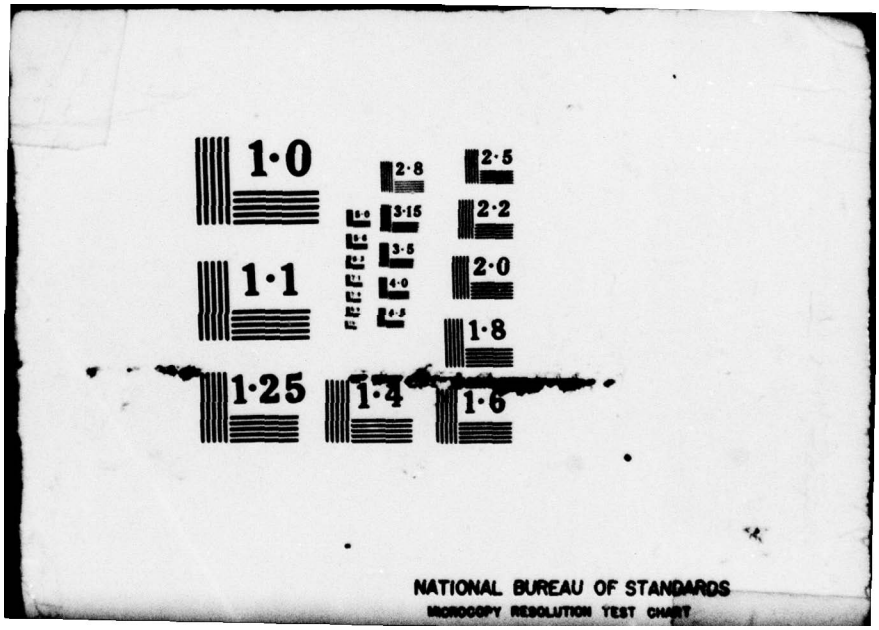
These factors will tend to reduce the execution time of a microprocessor implementation so that it will not impact significantly on the overall MSMS execution time.

Storage, input/output, or timing requirements do not present a significant problem to a microprocessor implementation. Thus, it is feasible to implement the MSMS Equipment Configuration, Data Collection, and Parameter Calculations subtask on a microprocessor.

AD-A071 563 COMPUTER SCIENCES CORP FALLS CHURCH VA F/G 9/2
ATEC SOFTWARE CONVERSION STUDY.(U)
DEC 76 R BRAUNTIGAM, H REINHARDT, J STOHLER DCA100-76-C-0048
UNCLASSIFIED CSC-R4968-1-1 SBIE -AD-E100 236 NL

2 OF 2
AD
A071563

END
DATE
FILMED
8-79
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

SECTION 4 - SAMPLE CONVERSION AND DEMONSTRATION

To demonstrate the feasibility of code conversion from the H316 to a microprocessor, the segment of the PATE that performs DDMS Equipment Configuration and Data Collection was converted to a microprocessor, and simulated via a software simulator. The sample conversion makes it possible to compare the timing and storage requirements of a program on the H316 and the same program converted to microprocessor code and executed on a microprocessor. Section 3 recommended that the DDMS Equipment Configuration, Data Collection and Parameter Calculation module be implemented on one 16-bit microprocessor. This sample conversion and demonstration was limited to the DDMS Equipment Configuration and Data Collection routines since access was only available to an 8-bit microprocessor (Motorola M6800) and it was the Parameter Calculation routine that dictated the requirement for a 16-bit microprocessor. Since the Equipment Configuration and Data Collection routines could be implemented on either an 8 or 16-bit microprocessor these routines were selected for the sample conversion and demonstration.

This section describes the software to be converted, the proposed microcomputer-based system, the cross-assembler and simulator used for code development and testing, and the results obtained from the demonstration. Listings of the microprocessor code and the simulation execution results are included.

4.1 PATE SOFTWARE CONVERSION

The Equipment Configuration and Data Collection functions in DDMS were chosen for the sample conversion to microprocessor code for the following reasons:

- They are relatively short
- They involve very little arithmetic, and no floating point nor double precision (32-bit) arithmetic
- They have very little interaction with the data base

- In any system that includes microprocessors, these are the functions most likely to be performed on a microprocessor. Other functions which involve more "number crunching" and data base access, such as alarm analysis, trending, and report generation, would be done on a larger machine at a higher level.

- Equipment Configuration and Data Collection are easily isolated from the rest of the system and implemented as a separate module. This is not true of functions such as the Scan Sequencer or Operator Interaction, which interface heavily with other functions.

In the PATE DDMS, Equipment Configuration and Data Collection are implemented in subroutine DEDEMS. Figure 4-1 is a hierarchical diagram of the DEDEMS software and Figure 4-2 is a functional block diagram of the DEDEMS subroutine. The following paragraphs describe the processing performed by this subroutine.

When the Scan Sequencer has selected a channel to be monitored, it calls subroutine CHM1, which interfaces between the Scan Sequencer, data base, and DEDEMS. CHM1 sets up the Input Argument Block and calls DEDEMS.

DEDEMS calls IDIN to inhibit the Pulse to Digital Converter (PDC) interrupts and configures the equipment by sending command words to the signal processing hardware. These command words are determined using various values in the Input Argument Block. Subroutine TEST is called to ensure that the inputs are within specified ranges. If DEDEMS is called for a self-test, subroutine SETEDD would be called to set up the hardware for a self-test.

Once the hardware is configured, SPDC is called to clear the data buffer and start the PDC. It calls EDIN to enable the PDC interrupt after the PDC is started.

DEDEMS next calls FORTRAN routine ZAPP, which calculates the channel parameters based on the data received from the PDC. ZAPP utilizes SUPS, SAMPLE, and TRNFER, which are Assembly Language subroutines located in DEDEMS. These subroutines initiate the sampling process, get the next data sample, and move the resulting parameters to the output area.

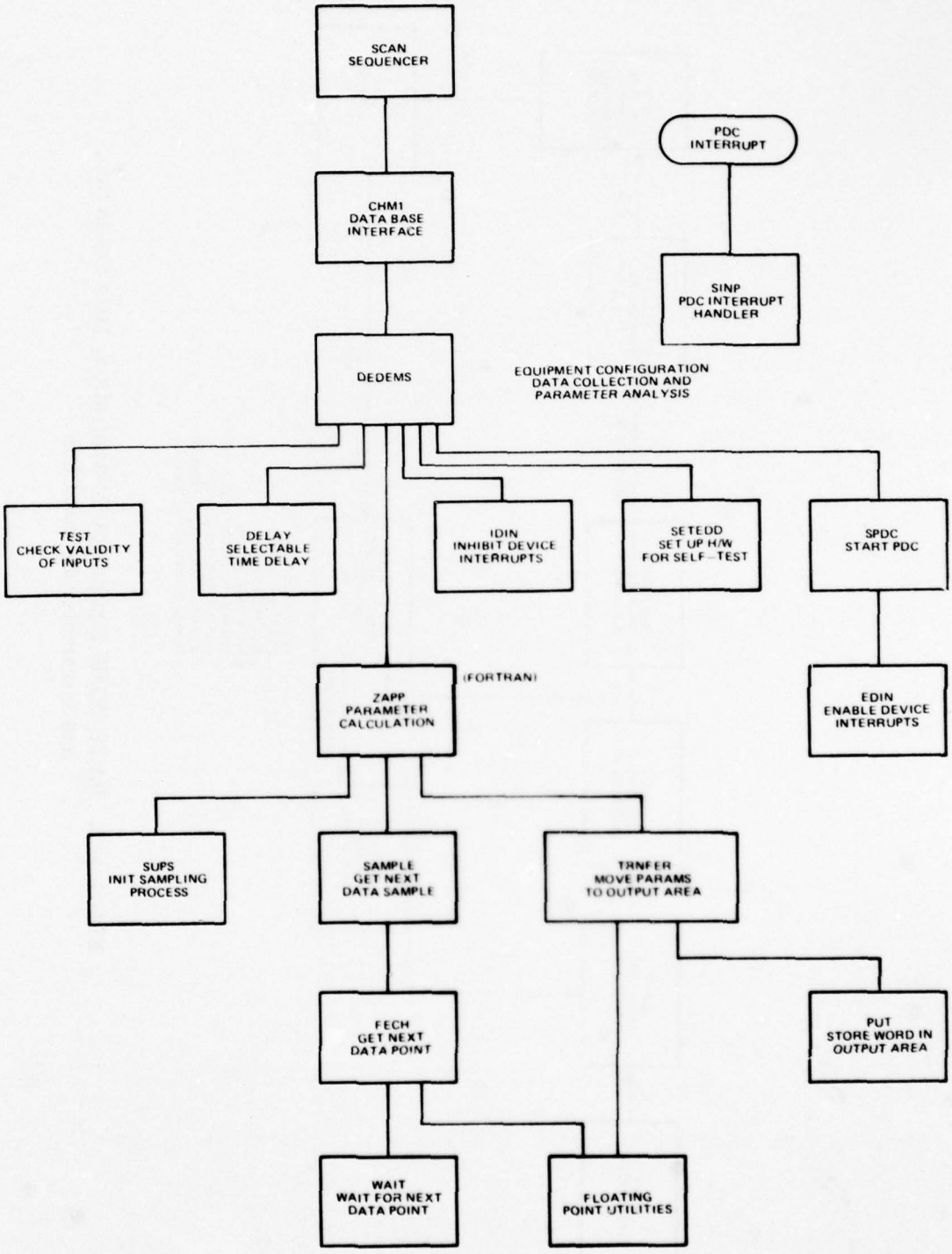


Figure 4-1. Hierarchical Diagram of DEDEMS Software Module

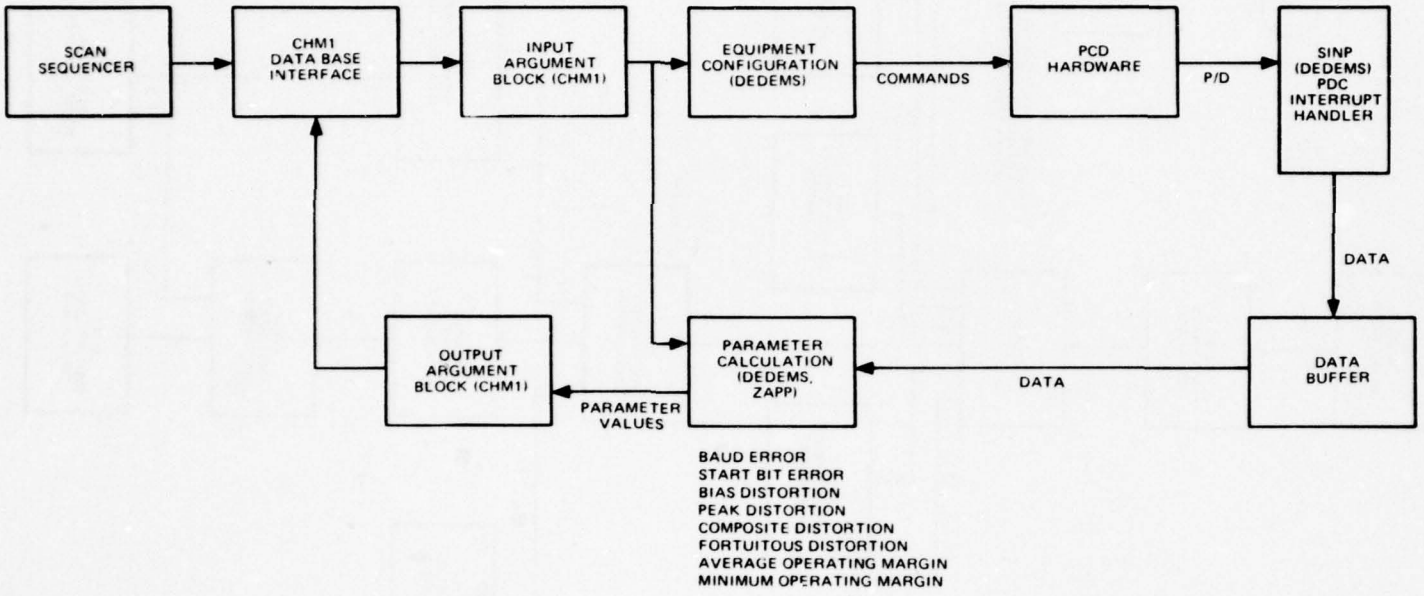


Figure 4-2. PATE DDMS Equipment Configuration, Data Collection, and Parameter Calculations

When a PDC interrupt occurs, control is passed to the interrupt service routine, SINP, located in DEDEMS. SINP inputs the Pulse/Digital (P/D) count from the PDC, and converts the counts to signal intervals, which are stored in the data buffer. ZAPP uses these signal intervals as data samples to compute parameters. When ZAPP has finished with a data sample, it waits until SINP stores a new data sample in the data buffer, and then performs calculations based on this new data sample. The data collection and parameter calculations are interleaved in this manner, until the data buffer becomes full. The results of ZAPP's parameter calculations are stored in the Output Argument Block, located in CHM1. They are used by other DDMS functions, such as Alarm Analysis, Trend Analysis, Channel Output, and Output Reports.

4.2 MICROPROCESSOR-BASED SYSTEM

To demonstrate the conversion of code from the H316 to a microprocessor, it is necessary to postulate a microprocessor environment, which includes both a microprocessor to execute the code and the interaction between that microprocessor and the outside world.

Figure 4-3 is a block diagram of a possible microprocessor-based system using a Motorola M6800 microprocessor. Characteristics of the M6800 are listed in Appendix B. The system might include an external computer acting as an ATEC control, which would perform data analysis, interact with the operator, and output results. It would also control a microprocessor which performs Equipment Configuration and Data Collection, based on inputs and commands received from the ATEC control. The software in the microprocessor would include Equipment Configuration, Control Interface, and PDC Interrupt Handler. The Control Interface enters inputs from the ATEC control into a local Input Argument block. The Equipment Configuration routine converts the input arguments to commands and sends them to the PDC hardware. The PDC Interrupt Handler inputs the P/D counts upon receiving an Interrupt from the PDC and converts the counts to data stored in the Data Buffer. The Control Interface sends the data from the Data Buffer to the ATEC Control for further analysis.

For purposes of the sample conversion/demonstration, the code corresponding to the Equipment Configuration and PDC Interrupt Handler were converted to M6800

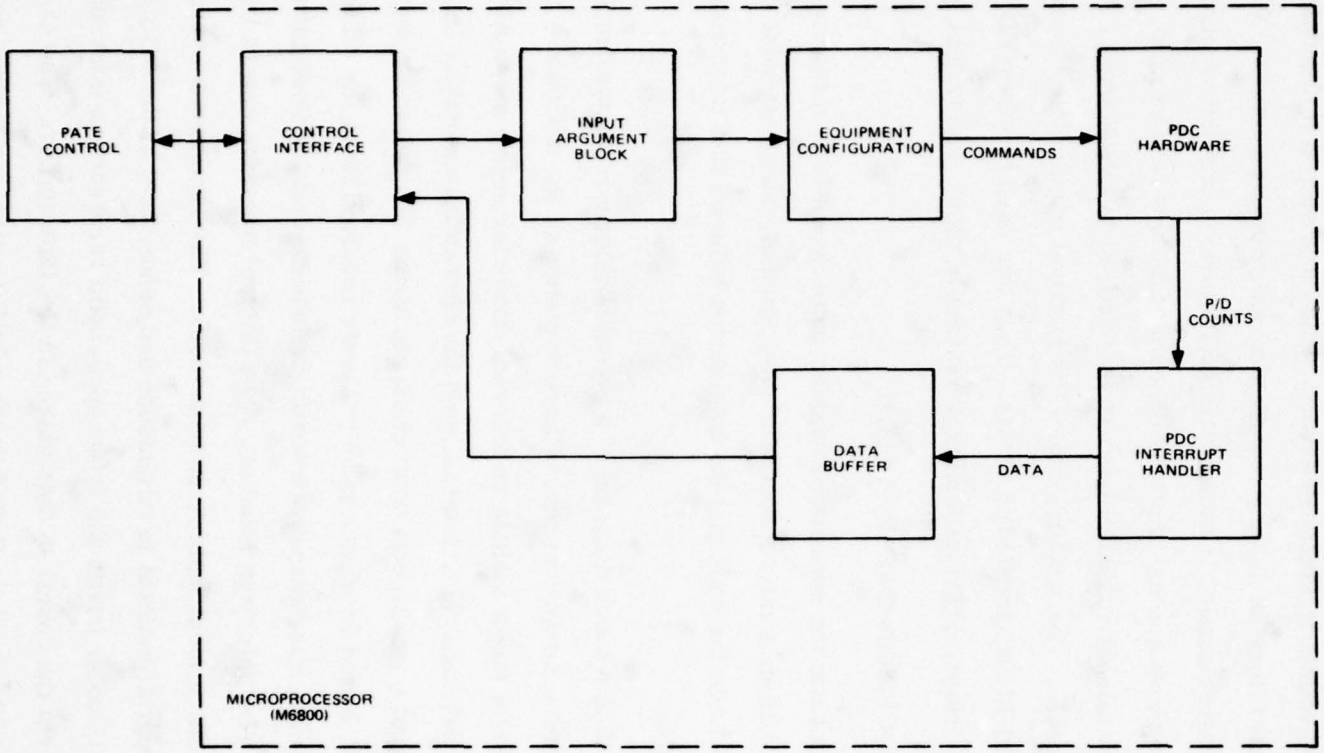


Figure 4-3. Possible Microprocessor-Based System

code. The Control Interface would depend largely on the exact nature of the external environment and the communications links to the M6800, neither of which are defined in the postulated system. Thus, the control interface is not included.

Since the hardware required to be configured and to supply P/D counts is not available, it is not practical to demonstrate the converted program in an actual M6800. For this reason, a Microtec M6800 software simulator running on a General Automation SPC-16 minicomputer was used to simulate the action of the hardware. This system provides:

- Listing command words to the hardware, so that they can be verified for correctness
- Providing interrupts to simulate the PDC
- Supplying P/D counts to the simulated program.

In addition, the simulator performs the following Control Interface functions:

- Provides inputs to Input Argument Block
- Initiates execution of simulated program
- Lists data buffer upon completion of execution to vierfy correct program operation.

The simulator also maintains a running total of the number of machine cycles used by the program being simulated. This enables accurate timing information to be determined from a simulation run. Figure 4-4 is a block diagram of the sample demonstration system.

4.3 MICROTEC M6800 CROSS-ASSEMBLER

The Microtec Cross-Assembler Program for the Motorola M6800 microprocessor was used to translate the assembly language code into hexadecimal object code. The assembler is written in ANSI Standard FORTRAN IV and is executed on a General Automation SPC-16 minicomputer.

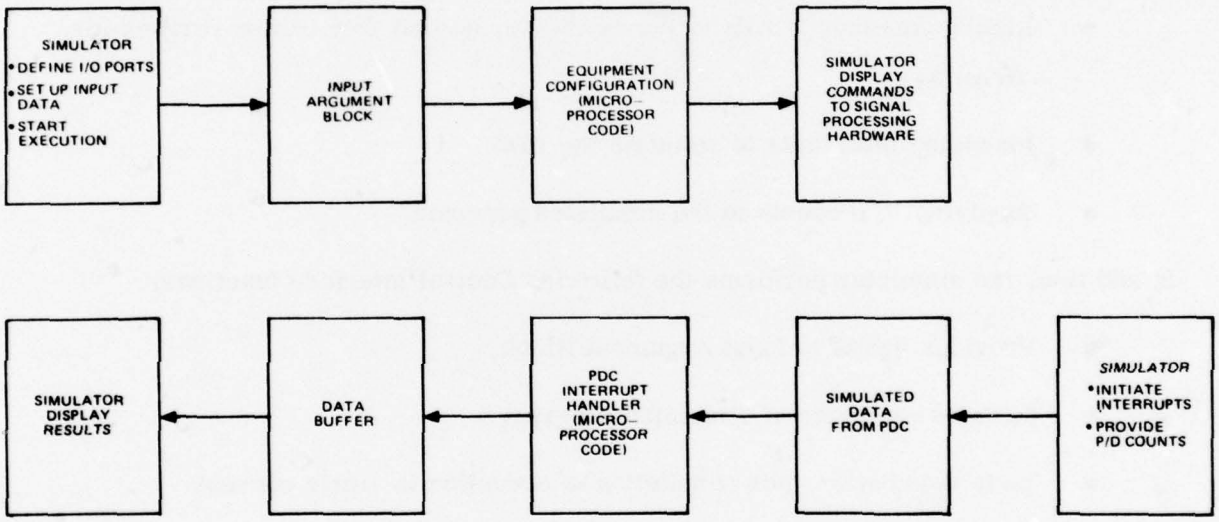


Figure 4-4. Sample Demonstration, Using Simulator Commands To Simulate PATE Control and PDC Hardware

The assembler is a two-pass program that includes macro capability, symbolic and relative addressing, forward references, expression evaluation, and data generation in octal, decimal, and hexadecimal number systems, as well as ASCII. The mnemonic operation codes as well as the pseudo-ops are identical to those used by Motorola in their literature and software products.

The assembler inputs a program in symbolic source code which has been previously prepared by the user. On the first pass it builds the symbol table. On the second pass it uses the symbol table to translate the program to machine code. It prints a complete listing (including both source code and object code in hexadecimal) as well as appropriate error messages and the symbol table. It also outputs the object code in a computer readable module.

4.4 MICROTEC M6800 SIMULATOR

The Microtec Software Simulator for the Motorola M6800 microprocessor interpretively executes any program assembled by the Microtec Motorola M6800 Cross-Assembler. It simulates ROM/RAM memory, input/output devices, interrupts, and M6800 registers and status bits. The simulator is written in ANSI standard FORTRAN IV and is executed on an General Automation SPC-16 minicomputer.

To simulate program execution, the user devises a sequence of commands in the Simulation Control Language (SCL), and enters the module and object modules produced by the cross-assembler and the SCL module into the simulator program. The simulator loads the object module into its simulated microprocessor memory and interpretively executes the program.

Through the commands given to the simulator, the user can specify start and stop addresses, declare memory locations as I/O ports, and set the initial contents of registers and memory locations. The user can also specify addresses which, when accessed either as instructions or as data, cause the simulator to print the state of the simulated registers, along with timing information, or the contents of a portion of memory. The simulator also has commands to simulate input data and interrupts.

The entire simulation run is documented for the user to study on a computer output listing. The listing displays the load module, list of simulator commands used, and all output generated during the simulation run as a result of the commands.

4.5 SAMPLE CONVERSION

The code within the DDMS routine DEDEMS that performs Equipment Configuration and Data Collection was converted to M6800 Assembly Language. The converted code was kept as close as possible to the original H316 code in structure and function. Figure 4-5 is a flow chart of the M6800 version of DEDEMS and Table 4-1 is a listing of the Assembly Language code.

When DEDEMS begins execution, it first initializes the stack pointer to point to the stack. It then inhibits interrupts by setting the interrupt enable flag. DEDEMS begins its equipment configuration phase by converting the channel number in the Input Argument Block (CHAN), to binary coded decimal (BCD), as required by the scanner. DEDEMS calls subroutine CHNSND to send the channel number to the scanner, two digits at a time. CHNSND fetches the two digits using the X-register, packs them into 1 byte, and send them to the scanner via the I/O port, SCAN.

DEDEMS next selects the signal type. It fetches the signal type from the Input Argument Block (TYPX) and calls subroutine TEST to determine if it is within the allowed limits. TEST compares the number in the A and B registers with the limits pointed to by the X-register. If the number is out of range, it is replaced with the closer limit. DEDEMS sends the signal type via the I/O port, SIGTYP.

DEDEMS next fetches the maximum allowed signal interval, MAXI, from the Input Argument Block and calls TEST to ensure that it is within the proper limits. It is then shifted left 5 bits and stored at MAXO. DEDEMS next reads two data words from the 10-MHz clock input port (CLK-IN) to clear the clock registers, and the clock is started by a one sent to the clock output port (CLKOUT). Equipment Configuration is now complete; subroutine SPDC is called to begin Data Collection.

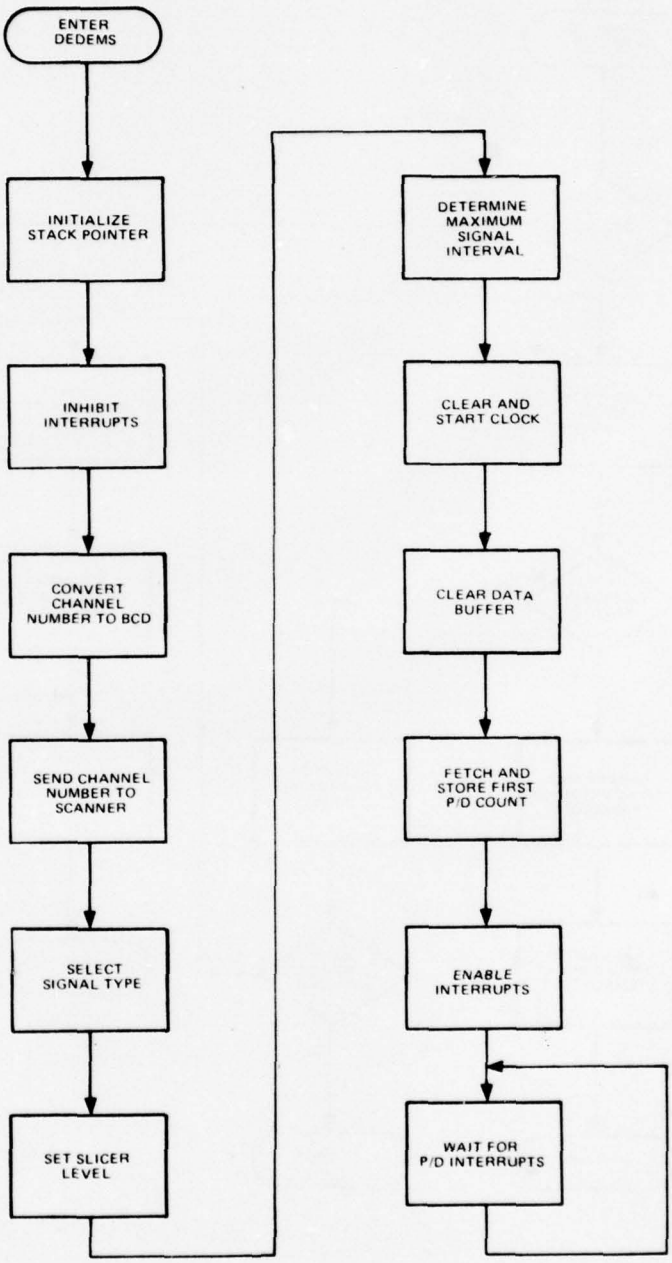


Figure 4-5. M6800 Version of DEDEMS (1 of 2)

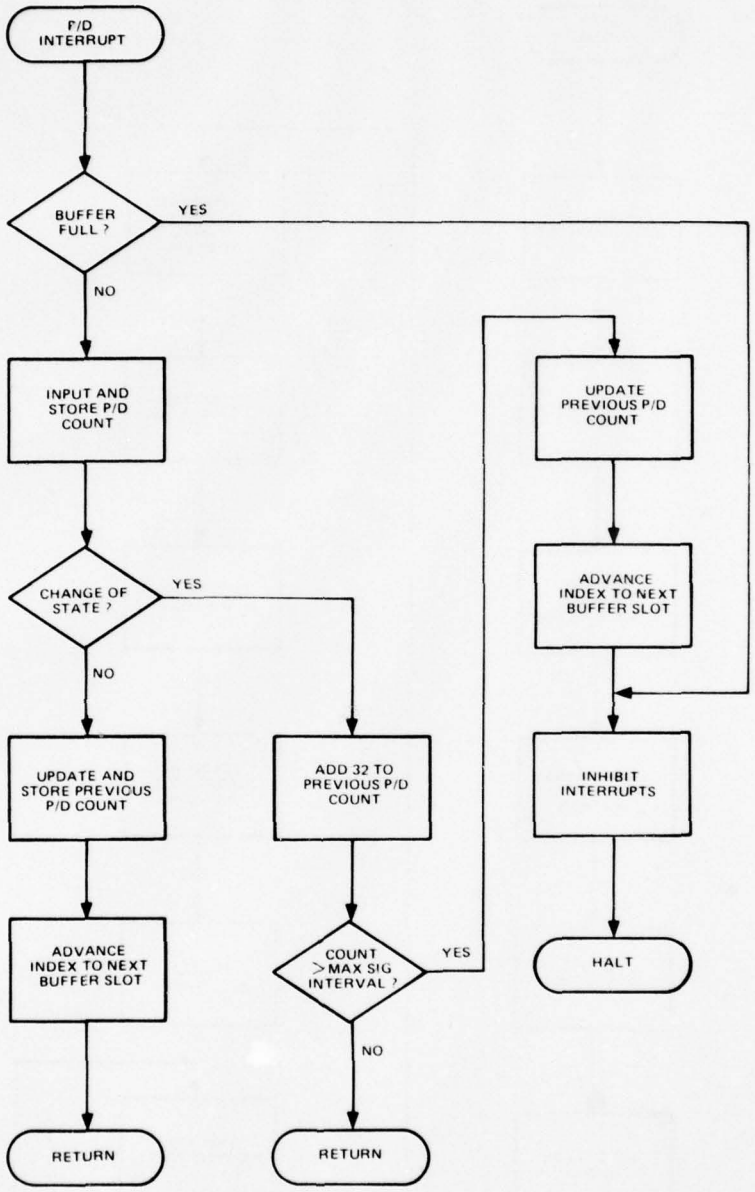


Figure 4-5. M6800 Version of DEDEMS (2 of 2)

Table 4-1. Assembly Listing (Sheet 1 of 8)

```
LINE    ADDR   B1  B2  B3 ERROR    DEDEMS
    1                                    TITLE   DEDEMS
    2                               *
    3                               * REVISION 8
    4                               *
    5                               * DEDEMS CONFIGURES THE SIGNAL PROCESSING HARDWARE FOR A
    6                               * SPECIFIC CHANNEL AND STARTS THE PDC.
    7                               * IT ALSO SERVICES INTERRUPTS FROM THE PDC,
    8                               * CONVERTS PDC COUNTS TO SIGNAL INTERVALS, AND
    9                               * STORES THEM IN THE DATA BUFFER
   10                               *
   11                                    ORG     0
   12   0000   8E  02  02                LDS     #STACK          SET UP STACK POINTER
   13   0003   00  03            DEDEMS  EQU     *
   14   0003   0F                        SEI                     INHIBIT INTERRUPTS
   15                               *
   16                               * CONVERT CHANNEL NUMBER TO BCD (4 DIGITS)
   17                               *
   18   0004   B6  01  45                LDA A   CHAN            A,B = CHANNEL NUMBER
   19   0007   F6  01  46                LDA B   CHAN+1
   20   000A   5C                        INC     B               ADD 1 TO CHANNEL NUMBER
   21   000B   89  00                    ADC A   #0
   22   000D   CE  01  5B                LDX     #DIGITS
   23   0010   FF  01  5F                STX     DPTR            DPTR POINTS TO BCD DIGITS
   24   0013   CE  01  53                LDX     #K1000          X POINTS TO CURRENT POWER OF TEN
   25   0016   00  16            LOOP1   EQU     *
   26   0016   7F  01  61                CLR     COUNT           INITIALIZE COUNTER
   27   0019   00  19            LOOP2   EQU     *
   28   0019   E0  01                    SUB B   1,X             SUBTRACT POWER OF TEN
   29   001B   A2  00                    SBC A   0,X
   30   001D   25  05                    BCS     OFLOW           EXIT LOOP IF OVERFLOW
   31   001F   7C  01  61                INC     COUNT           BUMP COUNT
   32   0022   20  F5                    BRA     LOOP2           CONTINUE LOOP
   33   0024   00  24            OFLOW   EQU     *
   34   0024   EB  01                    ADD B   1,X             RESTORE PARTIAL RESULT
   35   0026   A9  00                    ADC A   0,X
   36   0028   FF  01  62                STX     SAVEX           SAVE POINTER TO POWER OF TEN
   37   002B   FE  01  5F                LDX     DPTR            X = DIGIT POINTER
   38   002E   36                        PSH     A               SAVE A ON STACK
   39   002F   B6  01  61                LDA A   COUNT           MOVE COUNT TO DIGITS AREA
   40   0032   A7  00                    STA A   0,X
   41   0034   32                        PUL     A               RESTORE A
   42   0035   08                        INX                     ADVANCE DIGIT POINTER
   43   0036   FF  01  5F                STX     DPTR            UPDATE DIGIT POINTER
   44   0039   FE  01  62                LDX     SAVEX           RESTORE POINTER TO POWER OF TEN
   45   003C   08                        INX                     ADVANCE X TO NEXT POWER OF TEN
```

Table 4-1. Assembly Listing (Sheet 2 of 8)

```
 46   003D  08                          INX
 47   003E  8C  01  5B                  CPX     #K1000+8        DONE?
 48   0041  26  D3                     BNE     LOOP1           NO - CONTINUE LOOP
 49                                *
 50                                * SEND CHANNEL NUMBER TO HARDWARE
 51                                *
 52   0043  CE  01  5B                  LDX     #DIGITS         X POINTS TO DIGITS
 53   0046  BD  01  16                  JSR     CHNSND          SEND FIRST TWO DIGITS
 54   0049  08                          INX                     ADVANCE X TO NEXT TWO DIGITS
!LINE  ADDR  B1  B2  B3 ERROR      DEDEMS
 55   004A  08                          INX
 56   004B  BD  01  16                  JSR     CHNSND          SEND LAST TWO DIGITS
 57                                *
 58                                * SELECT SIGNAL TYPE
 59                                *
 60   004E  F6  01  47                  LDA B   TYPX            A,B = SIGNAL TYPE INDEX
 61   0051  4F                          CLR     A
 62   0052  CE  01  4B                  LDX     #TYPR           X = ADDRESS OF LIMITS
 63   0055  BD  00  F8                  JSR     TEST            TEST RANGE OF SIGNAL TYPE INDEX
 64   0058  F7  01  9A                  STA B   SIGTYP          OUTPUT SIGNAL TYPE
 65                                *
 66                                * SET SLICER LEVEL
 67                                *
 68   005B  B6  01  4A                  LDA A   SLIC            A = SLICER SETTING
 69   005E  B7  01  99                  STA A   SLICER          SET SLICER LEVEL
 70                                *
 71                                * FETCH MAXIMUM SIGNAL INTERVAL
 72                                *
 73   0061  B6  01  48                  LDA A   MAXI            A,B = MAX SIGNAL INTERVAL
 74   0064  F6  01  49                  LDA B   MAXI+1
 75   0067  CE  01  4F                  LDX     #INTR           X = ADDRESS OF LIMITS
 76   006A  BD  00  F8                  JSR     TEST            TEST RANGE OF MAX SIGNAL INTERVAL
 77   006D  CE  00  05                  LDX     #5              SHIFT A,B LEFT 5 BITS
 78   0070  00  70                SHLOOP EQU     *
 79   0070  58                          ASL     B
 80   0071  49                          ROL     A
 81   0072  09                          DEX
 82   0073  26  FB                      BNE     SHLOOP
 83   0075  B7  01  64                  STA A   MAXO            MAXO = MAX SIGNAL INTERVAL
 84   0078  F7  01  65                  STA B   MAXO+1
 85                                *
 86                                * START PDC
 87                                *
 88   0078  B6  01  9C                  LDA A   CLKIN           READ FROM CLOCK TO CLEAR GARBAGE
```

Table 4-1. Assembly Listing (Sheet 3 of 8)

```
 89   007E  02                        NOP
 90   007F  B6  01  9C                LDA A   CLKIN           READ AGAIN FROM CLOCK
 91   0082  02                        NOP
 92   0083  86  01                    LDA A   #1
 93   0085  B7  01  9B                STA A   CLKOUT          PULSE CLOCK
 94   0088  BD  01  22                JSR     SPDC            START PDC
 95                                *
 96                                * EQUIPMENT ALL CONFIGURED
 97                                *
 98   008B  3E                    W   WAI                     WAIT FOR INTERRUPT
 99   008C  20  FD                    BRA     W
100                                *
101                                * HANDLE INTERRUPTS FROM PDC
102                                *
103   008E  00  8E                SINP    EQU     *
104   008E  FE  01  68                LDX     DATA            CURRENT BUFFER ADDRESS
105   0091  BC  01  6A                CPX     ENDA            END OF BUFFER ADDRESS
106   0094  27  60                    BEQ     STOP            BUFFER IS FULL
107   0096  B6  01  9D                LDA A   PDPORT          GET SIMULATED UPPER P/D COUNT
108   0099  F6  01  67                LDA B   Q               GET PREVIOUS UPPER P/D FROM Q
LINE  ADDR  B1  B2  B3 ERROR      DEDEMS
109   009C  B7  01  67                STA A   Q               PUT PRESENT UPPER P/D INTO Q
110   009F  F7  01  66                STA B   Q-1             PUT PREVIOUS UPPER P/D INTO Q-1
111   00A2  B6  01  9D                LDA A   PDPORT          GET SIMULATED MSB OF LOWER P/D COUNT
112   00A5  A7  06                    STA A   6,X             STORE MSB IN BUFFER
113   00A7  F6  01  9D                LDA B   PDPORT          GET SIMULATED LSB OF LOWER P/D COUNT
114   00AA  E7  07                    STA B   7,X             STORE LSB IN BUFFER
115   00AC  A8  02                    EOR A   2,X             CHECK FOR SIGNAL STATE CHANGE
116   00AE  2A  15                    BPL     SAMS            NO CHANGE
117   00B0  A6  00                    LDA A   0,X             STATE HAS CHANGED
118   00B2  E6  01                    LDA B   1,X
119   00B4  FB  01  66                ADD B   Q-1             DO FINAL UPDATE ON PREVIOUS UPPER P/D COU
120   00B7  89  00                    ADC A   #0
121   00B9  A7  00                    STA A   0,X             PUT PREVIOUS UPPER P/D INTO BUFFER
122   00BB  E7  01                    STA B   1,X
123   00BD  08                        INX                     INCREMENT INDEX TO NEXT BUFFER SLOT
124   00BE  08                        INX
125   00BF  08                        INX
126   00C0  08                        INX
127   00C1  FF  01  68                STX     DATA
128   00C4  3B                        RTI                     RETURN TO INTERRUPT POINT
129                                *
130                                * SIGNAL STATE HAS NOT CHANGED
131                                *
132   00C5  00  C5                SAMS    EQU     *
```

Table 4-1. Assembly Listing (Sheet 4 of 8)

```
 133   00C5  A6  00                    LDA A   0,X             PREVIOUS UPPER P/D
 134   00C7  E6  01                    LDA B   1,X
 135   00C9  CB  20                    ADD B   #@40            SET OVERFLOW BIT
 136   00CB  89  00                    ADC A   #0
 137   00CD  A7  00                    STA A   0,X             STORE UPDATED PREVIOUS UPPER P/D
 138   00CF  E7  01                    STA B   1,X
 139   00D1  F0  01  65                SUB B   MAXO+1          SUBTRACT MAX OVF ALLOWED (OCT 40)
 140   00D4  B2  01  64                SBC A   MAXO
 141   00D7  2A  09                    BPL     HALT            OVF > MAX ALLOWED
 142   00D9  A6  06                    LDA A   6,X             PRESENT LPD MSB
 143   00DB  E6  07                    LDA B   7,X             PRESENT LPD LSB
 144   00DD  A7  02                    STA A   2,X             STORE IN PREVIOUS LPD MSB
 145   00DF  E7  03                    STA B   3,X             STORE IN PREVIOUS LPD LSB
 146   00E1  3B                        RTI                     RETURN TO INTERRUPT POINT
 147   00E2  00  E2            HALT    EQU     *
 148   00E2  A6  00                    LDA A   0,X             PREVIOUS UPPER P/D
 149   00E4  E6  01                    LDA B   1,X
 150   00E6  FB  01  66                ADD B   0-1             ACCUMULATE UPPER P/D COUNT
 151   00E9  89  00                    ADC A   #0
 152   00EB  A7  00                    STA A   0,X             AND UPDATE BUFFER
 153   00ED  E7  01                    STA B   1,X
 154   00EF  08                        INX                     INCREMENT INDEX TO NEXT BUFFER SLOT
 155   00F0  08                        INX
 156   00F1  08                        INX
 157   00F2  08                        INX
 158   00F3  FF  01  68                STX     DATA
 159                           *
 160                           * INHIBIT FURTHER PDC INTERRUPT PROCESSING IF OVF OR BUFFER FULL
 161                           *
 162   00F6  00  F6            STOP    EQU     *
1LINE  ADDR  B1  B2  B3 ERROR    DEDEMS
 163   00F6  0F                        SEI                     SET INTERRUPT MASK - OVF OR BUFFER FULL
 164   00F7  3E                        WAI                     HALT PROGRAM
 165                           *
 166                           * TEST ALGEBRAICALLY COMPARES THE CONTENTS OF A,B REGISTERS
 167                           * WITH RANGE LIMITS TO CHECK VALIDITY.
 168                           * IF CONTENTS ARE OUT OF RANGE, TEST SUBSTITUTES
 169                           * THE CORRESPONDING LIMIT.
 170                           *
 171                           * CALLING SEQUENCE:
 172                           *       X = ADDRESS OF LIMITS
 173                           *       A,B = NUMBER TO BE TESTED (A = HIGH ORDER BYTE)
 174                           *       JSR     TEST
 175                           *       RETURN
 176                           *
```

Table 4-1. Assembly Listing (Sheet 5 of 8)

```
177    00F8   00  F8            TEST   EQU    *
178                              *
179                              * TEST LOWER LIMIT
180                              *
181    00F8   A1  00                   CMP A  0,X          COMPARE HIGH ORDER BYTES
182    00FA   2E  0B                   BGT    UPLIM        OK - TEST UPPER LIMIT
183    00FC   2D  04                   BLT    LOWSUB       OUT OF RANGE
184    00FE   E1  01                   CMP B  1,X          COMPARE LOW ORDER BYTES
185    0100   2C  05                   BGE    UPLIM        OK - TEST UPPER LIMIT
186    0102   01  02            LOWSUB EQU    *
187    0102   A6  00                   LDA A  0,X          SUBSTITUTE LOWER LIMIT
188    0104   E6  01                   LDA B  1,X
189    0106   39                       RTS
190                              *
191                              * TEST UPPER LIMIT
192                              *
193    0107   01  07            UPLIM  EQU    *
194    0107   A1  02                   CMP A  2,X          COMPARE HIGH ORDER  BYTE
195    0109   2D  0A                   BLT    RETURN       OK - RETURN
196    010B   2E  04                   BGT    HISUB        OUT OF RANGE
197    010D   E1  03                   CMP B  3,X          COMPARE LOW ORDER BYTE
198    010F   2F  04                   BLE    RETURN       OK - RETURN
200    0111   01  11            HISUB  EQU    *
201    0111   A6  02                   LDA A  2,X          SUBSTITUTE HIGHER LIMIT
202    0113   E6  03                   LDA B  3,X
203    0115   01  15            RETURN EQU    *
204    0115   39                       RTS                 RETURN
205                              *
206                              * SEND TWO DIGITS OF CHANNEL NUMBER TO HARDWARE
207                              * X POINTS TO THE TWO DIGITS
208                              *
209    0116   01  16            CHNSND EQU    *
210    0116   A6  00                   LDA A  0,X          A = FIRST DIGIT
211    0118   48                       ASL    A
212    0119   48                       ASL    A
213    011A   48                       ASL    A
214    011B   48                       ASL    A
215    011C   AB  01                   ADD A  1,X          PACK SECOND DIGIT INTO A
1LINE  ADDR   B1  B2  B3 ERROR    DEDEMS
216    011E   B7  01  98                STA A  SCAN         SET UP CHANNEL NUMBER IN SCANNER
217    0121   39                       RTS                 RETURN
218                              *
219                              * SPDC STARTS THE PDC AND INPUTS THE FIRST POINT
220                              *
221    0122   01  22            SPDC   EQU    *
```

Table 4-1. Assembly Listing (Sheet 6 of 8)

```
222  0122  CE 01 6C          LDX     #BUFADR         X = ADDRESS OF BUFFER
223  0125  01 25       ZLOOP  EQU     *
224  0125  6F 00              CLR     0,X             CLEAR BUFFER LOCATION
225  0127  08                 INX                     ADVANCE TO NEXT BUFFER LOCATION
226  0128  8C 01 94            CPX     #BUFEND         END OF BUFFER?
227  012B  26 F8              BNE     ZLOOP           NO - CONTINUE LOOP
228                       *
229                       * END OF LOOP
230                       *
231  012D  CE 01 6C          LDX     #BUFADR
232  0130  FF 01 68          STX     DATA            DATA POINTS TO START OF BUFFER
233                       *
234                       * FETCH FIRST DATA POINT
235                       *
236  0133  B6 01 9D          LDA A   PDPORT
237  0136  B7 01 67          STA A   Q               HIGH ORDER WORD
238  0139  B6 01 9D          LDA A   PDPORT
239  013C  A7 02             STA A   2,X             LOW ORDER WORD
240  013E  B6 01 9D          LDA A   PDPORT
241  0141  A7 03             STA A   3,X
242  0143  0E                CLI                     CLEAR INTERRUPT MASK ENABLE INTERRUPTS
243  0144  39                RTS                     RETURN
244                       *
245                       * CONSTANTS AND VARIABLES
246                       *
247                       * INPUT ARGUMENT BLOCK
248                       *
249  0145  00 0A       CHAN   FDB     10              CHANNEL NUMBER
250  0147  01          TYPX   DATA    1               SIGNAL TYPE INDEX
251  0148  00 0F       MAXI   FDB     15              MAXIMUM SIGNAL INTERVAL
252  014A  14          SLIC   FCB     20              SLICER SETTING
253                       *
254                       * RANGE LIMITS
255                       *
256  014B  00 00       TYPR   FDB     0               TYPX
257  014D  00 02              FDB     2
258  014F  00 01       INTR   FDB     1               100 MS
259  0151  00 64              FDB     100             100 SECONDS
260                       *
261                       * VARIABLES FOR BINARY TO BCD CONVERSION
262                       *
263  0153  03 E8       K1000  FDB     1000            POWERS OF TEN
264  0155  00 64              FDB     100
265  0157  00 0A              FDB     10
266  0159  00 01              FDB     1
```

Table 4-1. Assembly Listing (Sheet 7 of 8)

```
 267   015D                          DIGITS RES    4          STORAGE FOR DIGITS OF CHANNEL NUMBER
 268   015F                          DPTR   RES    2          POINTER TO CURRENT DIGIT
 269   0161                          COUNT  RES    1          COUNT UP TO CURRENT DIGIT
LINE   ADDR   B1  B2  B3 ERROR        DEDEMS
 270   0162                          SAVEX  RES    2          TEMP STORAGE FOR X REGISTER
 271                                 *
 272                                 * OTHER VARIABLES
 273                                 *
 274   0164                          MAXO   RES    2          MAXIMUM SIGNAL INTERVAL
 275   0166                                 RES    1          TEMP STORAGE FOR PREVIOUS UPPER P/D
 276   0167                          Q      RES    1          TEMP STORAGE FOR CURRENT UPPER P/D
 277   0168  01  6C                DATA   ACON   BUFADR      ADDRESS OF START OF BUFFER
 278   016A  01  94                ENDA   ACON   BUFEND      ADDRESS OF END OF BUFFER
 279   016C  01  6C                BUFADR EQU    *
 280   016C                                 RES    40         DATA BUFFER
 281   0194  01  94                BUFEND EQU    *
 282   0194                                 RES    4          PADDING FOR BUFFER OVERRUN
 283                                 *
 284                                 * INPUT/OUTPUT PORTS
 285                                 *
 286   0198                          SCAN   RES    1          OUTPUT PORT TO SCANNER
 287   0199                          SLICER RES    1          OUTPUT PORT TO SLICER
 288   019A                          SIGTYP RES    1          OUTPUT PORT TO SIGNAL TYPE
 289   019B                          CLKOUT RES    1          OUTPUT PORT TO CLOCK
 290   019C                          CLKIN  RES    1          INPUT PORT FROM CLOCK
 291   019D                          PDPORT RES    1          INPUT PORT FROM P/D
 292                                 *
 293                                 * STACK SPACE
 294                                 *
 295   019E                                 RES    100
 296   0202                          STACK  RES    1
 297                                 *
 298                                 * SET UP INTERRUPT VECTOR
 299                                 *
 300                                       ORG    2047-7     (ASSUMING 2K OF MEMORY)
 301   07F8  00  8E                        ACON   SINP
 302   07FA                                END
  TOTAL ASSEMBLER ERRORS =    0
```

Table 4-1. Assembly Listing (Sheet 8 of 8)

```
                                DEDEMS
                            SYMBOL TABLE

* 1
BUFADR  016C    BUFEND  0194    CHAN    0145    CHNSND  0116
CLKIN   019C    CLKOUT  019B    COUNT   0161    DATA    0168
DEDEMS  0003    DIGITS  015B    DPTR    015F    ENDA    016A
HALT    00E2    HISUB   0111    INTR    014F    K1000   0153
LOOP1   0016    LOOP2   0019    LOWSUB  0102    MAX(    0148
MAXO    0164    OFLOW   0024    PDPORT  019D    Q       0167
RETURN  0115    SAMS    0005    SAVEX   0162    SCAN    0198
SHLOOP  0070    SIGTYP  019A    SINP    008E    SLIC    014A
SLICER  0199    SPDC    0122    STACK   0202    STOP    00F6
TEST    00F8    TYPR    014B    TYPX    0147    UPLIM   0107
W       008B    ZLOOP   0125
1
```

SPDC clears the data buffer to zero and inputs the first data point from the PDC. Each data point consists of a 20-bit count, an overflow bit, and a signal state bit, arranged in 3 bytes as illustrated in Figure 4-6. The data used for parameter calculations consist of intervals between signal state transitions. The PDC counts thus have to be converted to intervals. One interval may include several PDC counts; thus, the intervals are stored in 4 bytes in the data buffer while the PDC counts occupy only 3 bytes.

The PDC has a count available when either the signal state changes or the count reaches its maximum value. After the initial read, if there is no change of state, the maximum count (20_{16}) is added to the data buffer location corresponding to the high order byte from the PDC. The low order 2 bytes are stored in the data buffer, and the count received is saved. When a change of state is detected, the previous high order word is added to the corresponding memory location, the current input from the PDC is saved, and the pointer to the data buffer is advanced to the next 4-byte slot.

The first byte SPDC reads from the PDC is stored in location Q, and the next 2 bytes are stored in the data buffer in the locations corresponding to the least significant 2 bytes of the first data point. Then the interrupt is enabled so that the PDC can interrupt when it has a count ready, and SPDC returns. DEDEMS next enters a loop in which it waits for interrupts from the PDC.

When an interrupt is received from the PDC, control is passed to SINP, the PDC service routine. SINP first tests the pointer to the data buffer to determine if the buffer is full; if so, SINP disables the PDC interrupt and exits. Otherwise, the first byte from the PDC is input and stored in location Q. The next 2 bytes are input and stored in the data buffer, and the previous contents of Q are moved to location Q-1. The signal state bit of the current input is compared with the signal state bit of the previous input to determine if the state has changed. If it has, Q-1, which contains the high order byte of the previous count, is added to the corresponding location in the data buffer, and the pointer to the data buffer is advanced to the next 4-byte slot.

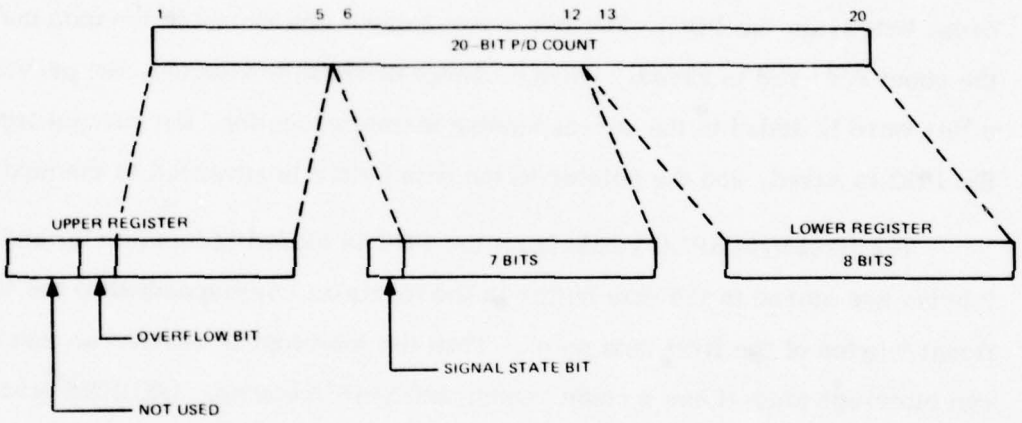


Figure 4-6. PDC Input Word Format

If the state did not change, the maximum count (20_{16}) is added to the data buffer location corresponding to the high order byte, and the sum is compared with MAXO, the maximum allowed signal interval. If the sum exceeds MAXO, the data is treated as though there had been a change of state. Q-1 is added to the data buffer and the data buffer pointer is advanced. The program then halts.

DEDEMS continues in this manner until either the data buffer becomes full or a MAXO is exceeded, at which point the interrupt is disabled and the program enters the wait state, halting the program.

4.6 ASSEMBLY LISTING

Table 4-1 is the listing of the code converted to M6800 Assembly Language. The first column is the line number of each source code statement. The second column is the address at which the corresponding instruction was assembled. The next three columns contain the object code corresponding to each instruction. The address and the object code are in hexadecimal notation. Note that each instruction assembles into 1 to 3 bytes.

Following the object code is the source code which includes labels, instructions, operands, pseudo-ops, and comments. At the end of the listing is the symbol table, which has all labels used with the corresponding address in hexadecimal.

4.7 M6800 SIMULATION

The Microtec M6800 Simulator was used to verify the operation of the Equipment Configuration and Data Collection code which was converted to M6800 microprocessor language. The simulator has two inputs, a binary object module and a list of commands. The binary input to the simulator is the object module generated by the M6800 assembler. The command input prepared by the user and stored in the DS (SIMCOM) file controls the operation and provides the input to the simulator. All output was printed on the line printer. Table 4-2 is a listing of the simulator output.

Table 4-2. Simulator Output Listing (Sheet 1 of 3)

```
1 MICROTEC 6800 SIMULATOR                    VERSION 1.0
                        OBJECT MODULE
S00600004844521B
S11F0000E02020FB60145F60146508900CE015DFF015FCE01537F0161E001A212
S11F0010C0025057C016120F5EB01A900FF0162FE015F36B60161A70032C8FF0123
S11F0038FFE016208088C015B26D3CE015BBD0116080BBD0116F601474FCE01B9
S11F00544BBD00F8F7019AB6014AB70199B6014BF6014ACE014FBD00F8C5C005C3
S11F0070584907266DB70164F70165B6019C02B6019C028601B7019DBD01223E8A
S11F008C20FDFE0163BC016A2760B6019DF60167B70167F70166B6019DA706F6FC
S11F00A8019DE707A8022A15A600E601FB01668900A700E701080808FF016834
S11F00C48A600E601CD208900A700E701F00165B201642A09A606E607A702E783
S11F00E0033BA600E601FB01668900A700E701080808FF01680F3EA100ZE0B0C
S11F00FC2004E1012C05A600E60139A1022D0A2E04E1032F04A602E60339A60047
S11F011C482484348AB01B7019839CE016C6F00088C019426F8CE016CFF0168B628
S11F013401 9DB70167B6019DA702B6019DA7030E39000A01000F140000002007C
S10E015001006403E80064000A0001E1
S1070168016C01948D
S1050F8008E6D
S9030000FC
1 MICROTEC 6800 SIMULATOR                    VERSION 1.0
 **  DS(SIMCOM)
 **  THIS IS A BATCH COMMAND STREAM TO THE 6800 SIMULATOR
 **  IT IS INTENDED TO TEST DS(DEDEMS), REVISION 8
ENDM 7FF
START 0
STOP F7
INTA 8B I
SETIO 198 199 19A 19B 19C 19D
ADDR  198 199 19A 19B 19C 19D
DUMP F6 145 19D
DATA 17 18
DATA 00 00 01 00 00 02 00 00 03 00 80 04 00 00 05
DATA 00 80 06 00 80 07 00 80 08 00 00 09 00 80 0A
DATA 00 00 0B 00 80 0C 00 00 0D 00 80 0E 00 00 0F
FEND
```

Table 4-2. Simulator Output Listing (Sheet 2 of 3)

```
1 MICROTEC 6800 SIMULATOR                          VERSION 1.0
0 PC      INST          EA     (EA)    NPC     HINZVC    A    B     X     SP    CYCLE
 011E    STAA 0198     0198     IO     0121    010100    00   00   015B  0200   000468
 011E    STAA 0198     0198     IO     0121    010000    11   00   015B  0200   000513
 0058    STAB 019A     019A     IO     005B    010001    00   01   014B  0202   000590
 005E    STAA 0199     0199     IO     0061    010001    14   01   014B  0202   000599
 007B    LDAA 019C     019C     IO     007E    010000    17   E0   0000  0202   000745
 007F    LDAA 019C     019C     IO     0082    010000    18   E0   0000  0202   000751
 0085    STAA 019B     019B     IO     0088    010000    01   E0   0000  0202   000760
 0133    LDAA 019D     019D     IO     0136    010100    00   E0   016C  0200   001505
 0139    LDAA 019D     019D     IO     013C    010100    00   E0   016C  0200   001514
 013E    LDAA 019D     019D     IO     0141    010000    01   E0   016C  0200   001524
 INTERRUPT RECOGNIZED      PC = 008B
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   016C  01FB   001568
 00A2    LDAA 019D     019D     IO     00A5    010100    00   00   016C  01FB   001586
 00A7    LDAB 019D     019D     IO     00AA    010000    00   02   016C  01FB   001596
 INTERRUPT RECOGNIZED      PC = 008B
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   016C  01FB   001716
 00A2    LDAA 019D     019D     IO     00A5    010100    00   00   016C  01FB   001734
 00A7    LDAB 019D     019D     IO     00AA    010000    00   03   016C  01FB   001744
 INTERRUPT RECOGNIZED      PC = 008B
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   016C  01FB   001864
 00A2    LDAA 019D     019D     IO     00A5    011000    80   00   016C  01FB   001882
 00A7    LDAB 019D     019D     IO     00AA    010000    80   04   016C  01FB   001892
 INTERRUPT RECOGNIZED      PC = 008B
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   0170  01FB   002002
 00A2    LDAA 019D     019D     IO     00A5    010100    00   00   0170  01FB   002020
 00A7    LDAB 019D     019D     IO     00AA    010000    00   05   0170  01FB   002030
 INTERRUPT RECOGNIZED      PC = 008B
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   0174  01FB   002140
 00A2    LDAA 019D     019D     IO     00A5    011000    80   00   0174  01FB   002158
 00A7    LDAB 019D     019D     IO     00AA    010000    80   06   0174  01FB   002168
 INTERRUPT RECOGNIZED      PC = 008B
0 PC      INST          EA     (EA)    NPC     HINZVC    A    B     X     SP    CYCLE
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   0178  01FB   002278
 00A2    LDAA 019D     019D     IO     00A5    011000    80   00   0178  01FB   002296
 00A7    LDAB 019D     019D     IO     00AA    010000    80   07   0178  01FB   002306
 INTERRUPT RECOGNIZED      PC = 008B
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   0178  01FB   002426
1 MICROTEC 6800 SIMULATOR                          VERSION 1.0                PAGE   4
 00A2    LDAA 019D     019D     IO     00A5    011000    80   00   0178  01FB   002444
 00A7    LDAB 019D     019D     IO     00AA    010000    80   08   0178  01FB   002454
 INTERRUPT RECOGNIZED      PC = 008B
 0096    LDAA 019D     019D     IO     0099    010100    00   E0   0178  01FB   002574
 00A2    LDAA 019D     019D     IO     00A5    010100    00   00   0178  01FB   002592
 00A7    LDAB 019D     019D     IO     00AA    010000    00   09   0178  01FB   002602
```

Table 4-2. Simulator Output Listing (Sheet 3 of 3)

```
 INTERRUPT RECOGNIZED     PC = 008B
 0096   LDAA 019D      019D    IO      0099    010100    00   E0   017C   01FB   002712
 00A2   LDAA 019D      019D    IO      00A5    011000    80   00   017C   01FB   002730
 00A7   LDAB 019D      019D    IO      00AA    010000    80   0A   017C   01FB   002740
 INTERRUPT RECOGNIZED     PC = 008B
 0096   LDAA 019D      019D    IO      0099    010100    00   E0   0180   01FB   002850
 00A2   LDAA 019D      019D    IO      00A5    010100    00   00   0180   01FB   002868
 00A7   LDAB 019D      019D    IO      00AA    010000    00   0B   0180   01FB   002878
 INTERRUPT RECOGNIZED     PC = 008B
 0096   LDAA 019D      019D    IO      0099    010100    00   E0   0184   01FB   002988
 00A2   LDAA 019D      019D    IO      00A5    011000    80   00   0184   01FB   003006
 00A7   LDAB 019D      019D    IO      00AA    010000    80   0C   0184   01FB   003016
 INTERRUPT RECOGNIZED     PC = 008B
 0096   LDAA 019D      019D    IO      0099    010100    00   E0   0188   01FB   003126
 00A2   LDAA 019D      019D    IO      00A5    010100    00   00   0188   01FB   003144
 00A7   LDAB 019D      019D    IO      00AA    010000    00   0D   0188   01FB   003154
 INTERRUPT RECOGNIZED     PC = 008B
 0096   LDAA 019D      019D    IO      0099    010100    00   E0   018C   01FB   003264
 00A2   LDAA 019D      019D    IO      00A5    011000    80   00   018C   01FB   003282
 00A7   LDAB 019D      019D    IO      00AA    010000    80   0E   018C   01FB   003292
 INTERRUPT RECOGNIZED     PC = 008B
 0096   LDAA 019D      019D    IO      0099    010100    00   E0   0190   01FB   003402
0 PC       INST          EA    (EA)    NPC     HINZVC    A    B     X      SP    CYCLE
 00A2   LDAA 019D      019D    IO      00A5    010100    00   00   0190   01FB   003420
 00A7   LDAB 019D      019D    IO      00AA    010000    00   0F   0190   01FB   003430
 INTERRUPT RECOGNIZED     PC = 008B
 DUMP     PC = 00F6
 0140     9D A7 03 0E 39 00 0A 01 00 0F 14 00 00 00 02 00
 0150     01 00 64 03 E8 00 64 00 0A 00 01 00 00 01 01 01
1 MICROTEC 6800 SIMULATOR                  VERSION 1.0
 0160     5F 01 01 59 01 E0 00 00 01 94 01 94 00 40 00 03
 0170     00 00 80 04 00 00 00 05 00 40 80 08 00 00 00 09
 0180     00 00 80 0A 00 00 00 0B 00 00 80 0C 00 00 00 0D
 0190     00 00 80 0E 00 00 00 0F I0 I0 I0 I0 I0 I0 00 00
 00F7   WAI                              00F8    010100    01   E0   0194   01F4   003547
1
```

The simulator first lists the object module in hexadecimal notation, with one record to each line. The header record starts with "S0" and the end-of-file record starts with "S9". Data records start with "S1", followed by the byte count (two digits), load address (four digits), up to 28 bytes of data (two digits each), and a two-digit checksum. The load address corresponds to the address field and the data bytes correspond to the object code fields of the Assembly Language listing.

Following the object module, the simulator lists the commands which control the simulation run. ENDM sets the highest memory location. This determines the M6800 interrupt vectors and when an instruction accesses a location outside of memory. START and STOP specify the start and stop addresses, respectively, for the current simulation. SETIO declares locations as I/O ports. When an instruction reads from an I/O port, a byte is supplied from the Input Buffer. Bytes are entered into the Input Buffer by the DATA commands.

INTA declares a location at which, when accessed, an interrupt is simulated. Also, the type of interrupt, either maskable or nonmaskable, is specified. The ADDR command directs the simulator to print the status of the simulated microprocessor whenever any of the addresses specified are accessed, whether as data or instructions. The status information consists of:

PC - Current Program Counter
INST - Instruction being Executed
EA - Effective Address
(EA) - Contents of Effective Address ("IO" if I/O Port)
NPC - Next Program Counter
HINZVC - Status Bits
A, B, X - Contents of Registers
SP - Contents of Stack Pointer
CYCLE - Number of Simulated Machine Cycles Elapsed Since Start of Simulation.

DUMP is similar to ADDR, except that when the first specified address is accessed, all the contents of memory from the second address to the third address are

printed. The command FEND signals the simulator that all the commands have been received and causes it to begin the simulation run.

In the simulation of the converted code, the memory size was established as 2048 bytes. The simulation starts execution at the beginning of the program, with equipment configuration, and stops when the interrupt service routine detects either an overflow condition or a buffer full condition. When a stop condition has been recognized, the constants and variables storage area of memory is dumped, from CHAN through PDPORT. This area also includes the data buffer. There are six locations declared to be I/O ports. These consist of output ports to the scanner, slicer, signal type output, and clock; and input ports from the clock and the PDC. An ADDR command is used to direct the simulator to print the status when any of the I/O ports are accessed.

When the simulated M6800 reads from an I/O port, the simulator supplies the next byte from its data buffer as input data to the M6800. The DATA command determines the contents of the data buffer. The first 2 bytes are used for the data read from the clock. The rest of the data are read in groups of 3 by the PDC interrupt handler and represent P/D counts. The P/D counts are in the format shown in Figure 4-5, with the most significant byte being read first. Interrupts are scheduled to occur after equipment configuration has been completed.

The output generated by the simulation run follows the simulator commands. In the simulation of the converted code, output is generated only when:

1. An I/O port is accessed (due to the ADDR command)
2. An interrupt is recognized (due to the INTA command)
3. DUMP address is reached at end of run
4. STOP address is reached at end of run.

The first two lines of output result from sending the channel number in BCD to the scanner (0011). The next line results from the signal type being set to one, followed by the output of the slicer level (14) to the slicer. Then the clock is read twice (17 and 18) and a pulse (1) is sent to the clock to start it. The next three lines correspond to

SPDC reading the first count from the PDC (00 00 01). This completes the Equipment Configuration.

The following output is due to interrupts and the interrupt handler reading the P/D counts in groups of 3 bytes. When the end of the buffer has been reached, the selected part of memory is dumped, the simulated program halts, and a final line is printed at the end of execution.

The part of the memory dump corresponding to the data buffer runs from 016C through 0193. Each slot consists of 4-bytes, and contains the total count between changes of signal state, as determined by the MSB of the second byte of each P/D count as illustrated in Figure 4-6. The data is structured to provide 10 intervals. The first interval requires three interrupts before the change of state as does the fourth interval. The other eight intervals change state in one interrupt. The two least significant bytes of each 4 byte slot come from the two least significant bytes of the last P/D count before the change of state. The two most significant bytes of each slot are 0020_{16} times the number of interrupts received without a change of state. Note that the two most significant bytes are 00 00 except for the first and fourth slots in which they are 00 40. These two slots correspond to the two three-interrupt intervals. For example, the first slot runs from 016C through 016F, which contain 00400003. This corresponds to the first three counts; 000001, 000002, and 000003. The 0003 comes from the last count, and the 0040 resulted where two P/D counts were received without the signal state changing. Each time this occurs, 200000 is added to the count. Thus, the count is 00200000 + 00200000 + 00000003 = 00400003 which is stored in the first 4-byte slot. It can be similarly verified that the data buffer contains the correct data corresponding to the inputs in the data commands. Note that the third byte in each 4-byte slot alternates between 80 and 00. This reflects the signal state bit, which alternates between one and zero in consecutive intervals.

4.8 COMPARISON OF RESULTS

The version of DEDEMS converted to M6800 code can be compared with the original version written for the H316 in terms of speed and storage requirements, and interface to other parts of DDMS. For the comparison to be meaningful, it must be assumed that both programs operate in the same mode and with the same inputs. The simulation has been described in Paragraph 4.7. To compare the microprocessor code with the corresponding H316 code, several assumptions must be made.

1. The H316 program has the same inputs, both in the Input Argument Block and from the PDC, as the M6800 program
2. DDMS is not operating in self-test mode
3. 100-millisecond delay is excluded from execution time
4. Input mode is via interrupts
5. Code in DEDEMS which calls ZAPP, and the subroutines called by ZAPP, are excluded from storage and timing estimates
6. There is no looping necessary for H316 I/O operations.

The simulation of the M6800 program required 3547 clock cycles. Assuming one microsecond per cycle, the M6800 requires 3.5 milliseconds to execute the program. This breaks down into two parts: 1.5 milliseconds for hardware configuration, and 2 milliseconds for data collection.

Using the instruction times given in Reference 3 the times required for the equipment configuration portion of the H316 code, which includes two subroutine calls to TEST, and calls to IDIN, SPDC, and EDIN, can be determined by adding the times for the instructions. The H316 equipment configuration code requires 0.58 milliseconds.

Similarly, the data collection time can be determined by adding the time required for the PDC interrupt service routine, SINP. SINP requires different amounts of time depending on the situation. If there is a change of signal state, SINP requires

91.2 microseconds. If there is no change of state, it requires 81.6 microseconds. The last interrupt, on which the end of the buffer is reached and data collection halts, requires 73.6 milliseconds. With the data given, there are four interrupts with no change of state, 10 with a change of state, and one at the end of the buffer. This gives a total of 1.31 milliseconds for data collection. These times are summarized in Table 4-3.

Overall, the M6800 requires 3.5 milliseconds to execute the program, and the H316 requires 1.89 milleseconds. Thus, the H316 is roughly a factor of two faster than the M6800. The difference in speed would be much greater if there were more arithmetic calculation, in which the 16-bit word length and the multiply, divide, and double precision hardware of the H316 could be used to best advantage.

Storage required consists of the code, variables, and constants used by the code, and the data buffer. Since the data buffer is common to both the M6800 and H316 versions of the code, and it takes up the same amount of storage in both, it is excluded from storage comparisons. The M6800 I/O ports are also excluded from the comparison because they do not occupy physical storage, only address space.

The M6800 requires 144_{16} bytes for program storage and 21_{16} bytes for constants and variables, giving a total of 165_{16} bytes, or, in decimal, 357 bytes. The H316 requires 266_{8} words for the storage of the corresponding code, which includes the equipment configuration portion of DEDEMS, subroutines TEST, SPDC, EDIN, and interrupt service routine SINP. This includes both code and variables and constants since it is difficult to separate the two areas of storage in the H316 software. This is 182 words in decimal, which is close to half the number of locations required by the M6800. Since the M6800 uses an 8-bit byte, and the H316 uses a 16-bit word, the memory requirements are almost exactly the same in the two machines.

The interface requirements for DEDEMS include interfaces to the various hardware devices and to the software modules. In the M6800, the hardware interface is via the six I/O ports. The hardware interface is via special I/O instructions in

Table 4-3. Comparison of Timing and Storage Between M6800 and H316

Timing	M6800 (Cycles)	M6800 (msec)	H316 (msec)
Equipment Configuration	1524	1.5	.58
Data Collection	2023	2.0	1.31
Total	3547	3.5	1.89
Storage			
M6800 (8-bit bytes)	357		
H316 (16-bit words)	182		
H316 (8-bit bytes)	364		

the H316. The H316 has more flexible I/O, in that the I/O instructions have the capability to loop until input data is ready or until output data is accepted by the device. Also, in the H316, individual interrupts can be inhibited, while in the M6800 there is only one interrupt mask bit for the machine. The H316 has another advantage in that it can input or output 16 bits at once, while the M6800 can only handle 8 bits.

The software interface consists of the Input Argument Block and the Data Buffer. In the H316, communication is simply through common core. The M6800 program would either have the software it communicates with in the same processor, in which case communication would be simple, or it would communicate with an external computer - either another microcomputer or a larger computer. In the latter case, additional software would be required to control the communications, and time would be spent receiving the input arguments and transmitting the data buffer at the conclusion of data collection.

In the sample demonstration it was found that:

1. The M6800 required twice the time as the H316 to execute the converted program
2. The M6800 required twice the number of storage locations as the H316. Since the H316 word size is twice that of the M6800, this amounts to the same amount of storage
3. The interface of the M6800 to the hardware environment is less flexible than the H316 interface
4. The interface of the M6800 to the software environment can be equivalent to the H316 or it can be more complex and time consuming, depending on the system architecture.

The code chosen for the conversion is well suited to a microprocessor. The M6800 would fare worse relative to the H316 on a program that involved more arithmetic, especially multiplication, division, and floating point operations.

SECTION 5 - SOFTWARE DEVELOPMENT COST ESTIMATE AND SCHEDULE

This section presents the estimated software development costs and recommended implementation schedule. The costs presented are those to design, code, debug, test, integrate and document the software recommended for conversion to microprocessor implementation for the IQCS, OQCS, DDMS and MSMS functions.

5.1 ASSUMPTIONS

The following assumptions were made in developing the estimated software development costs and implementation schedule.

- The same type of 16-bit microprocessor with hardware multiply and divide instructions will be used in each configuration
- The software will be written in assembly language
- Documentation will consist of a Program Maintenance Manual, Program Specification Manual, Test and Implementation Plan and Test Results
- Hardware and computer time costs are not included.

Section 3 described the requirements for a 16-bit microprocessor with hardware multiply and divide instructions. Microprocessors that meet these requirements include the National Semiconductor IMP16C/P, Texas Instrument TMS9900, Digital Equipment Corporation LSI11, Data General MicroNOVA, and General Automation 16/110. Characteristics of these microprocessors are listed in Appendix C. It is assumed that one of the above processors will be used throughout the system. This commonality of processors was assumed to reduce development and maintenance costs. Assembly language development has been assumed due to the unavailability of high level languages for some microprocessors and the lack of extensive use and testing for those high level language compilers that are available.

5.2 ESTIMATED COSTS

The software recommended for conversion to microprocessor implementation for the IQCS, OQCS, DDMS and MSMS functions was described in Section 3. Tables 5-1 to 5-8 identify the individual routines to be developed along with the estimated microprocessor lines of code for each routine. The combined I/OQCS function requires approximately 8957 lines of code, the DDMS, 2023, and the MSMS, 3379: a total of 14,359 lines of code. The cost to design, code, debug, test, integrate and document this code is estimated at $732,000. This includes $32,000 for hardware and software acquisition and checkout, $240,000 for design, $300,000 for development and $160,000 for integration and testing.

5.3 IMPLEMENTATION SCHEDULE

Figure 5-1 presents the implementation schedule. It is estimated that 18 calendar months will be required to complete the project. The task items identified in Figure-5-1 are those needed to assure successful project implementation. All documentation will be delivered according to DOD Automated Data System Documentation Standards Manual 4120, 17-M.

Table 5-1. I/OQCS Microprocessor A Software

NAME	FUNCTION	LINES OF CODE
ADR1	Inputs A/D data, squares, sums, and detects peak squared voltages	191
GACK	A/D gain adjustment	56
SMEC	Equipment configuration	100
UTILITIES*	I/O handlers and interrupt routines	250
TOTAL		597

*This software does not exist in the present PATE; it is peculiar to the microprocessor-based system.

Table 5-2. I/OQCS Microprocessor B Software

NAME	FUNCTION	LINES OF CODE
CAL1	Calculates AV, PI and maximum and minimum 10 and 50 millisecond average levels	155
CAL2	Calculates spectral width, noise width, frequency, area factor, frequency average, and parameter ratios PA, P1, P5, MI and MS	650
CAL3	Calculates weighted noise, flat noise, unweighted noise, noise frequency, and noise coloration, harmonic distortion, signal to noise ratio and VU	410
DBLO	dB calculation routine	89
FFT	Fast fourier transform algorithm	220
RSSM	Calculates the spectrum ratios R1, R2, R3 and R4	254
SMSQ	Unfolds FFT's, computes magnitude and accumulates results	172
SMTR	Traffic recognition	151
GENERAL	Format conversion routines such as binary to BCD, single to double precision, fixed to floating point, floating to fixed point	250
UTILITIES*	I/O handlers and interrupt routines	250
TOTAL		2601

*This software does not exist in the current PATE; it is peculiar to the microprocessor-based system.

Table 5-3. I/OQCS Microprocessor C Software

NAME	FUNCTION	LINES OF CODE
CHMO	Links channel selection, calculation, and traffic recognition routines	55
OTHER*	Additional software necessary to allow microprocessor C to exercise overall control of microprocessor-based system, e.g., subcommand decode, buffer control, overall timing control, etc.	200
UTILITIES*	I/O handlers and interrupt routines	250
		505

*This software does not exist in the present PATE; it is peculiar to the microprocessor-based system.

Table 5-4. I/OQCS Microprocessor D Software

NAME	FUNCTION	LINES OF CODE
COMIN*	Control of microprocessor/ higher level system communications line, i.e., buffering, error detection and correction, start and stop bit sensing, etc.	250
UTILITIES*	I/O handlers and interrupt routines	200
TOTAL		450

*This software does not exist in the current PATE; it is peculiar to the microprocessor-based system.

Table 5-5. I/OQCS Microprocessor E Software (Optional)

NAME	FUNCTION	LINES OF CODE
INOUT*	Decodes input from optional I/O device, communicates with higher level system, and displays output on I/O device	250
UTILITIES*	I/O handlers and interrupt routines	100
TOTAL		300

*This software is optional and will be needed only if local input and display capability is desired.

Table 5-6. I/OQCS Microprocessor F Software

NAME	FUNCTION	LINES OF CODE
1ATEST	Performs 1A test; calculates phase jitter, net loss, and frequency offset at 1000 Hz	95
1BTEST	Performs 1B test; calculates harmonic distortion and signal to noise ratio at 1000 Hz	156
1CTEST	Performs 1C test; counts phase hits, amplitude hits, and dropouts at 1000 Hz	330
26TEST	Performs 26 test; calculates phase jitter, net loss, and frequency offset at 2600 Hz	entry in 1ATEST*
AUTEST	Automatic test; calculates magnitude and delay versus frequency	470
CALC1	Converts raw input data to frequency (Hz), level (dBm x 10), and delay (microseconds)	143
CALC2	Converts raw input data to level (dBm x 10), frequency (Hz x 10), and phase jitter (deg.)	80
CALC3	Calculates level in fixed point for greater precision	43
DELY	Converts P/D number 1 counts to microseconds	19
DITEST	Performs dead idle test; C-MSG noise, 3-kHz flat noise, noise difference, noise coloration and noise frequency are calculated	entry in 1BTEST*
DT2KH	Detects 2000 Hz and transitions to 2100 Hz	128
FREQ	Converts P/D number 2 counts to frequency in Hz	43
GIMP	Obtains scanner addresses and line impedances	35
ICTIN	Reads impulse counter	33
IGAIN	Sets initial gain	187

Table 5-6. I/OQCS Microprocessor F Software (Cont'd)

NAME	FUNCTION	LINES OF CODE
INREAD	Impulse counter input	131
MNTEST	Manual test routine	284
NRMED	Normalizes envelope delay	329
NRMLV	Normalizes level	entry in NRMED*
OCHM	Initializes and calls IQCS FFT	40
OQST	Self test routine	78
RDTA	Inputs data from P/D counters for 1A and 1C tests	200
TESTCL	Test controller	532
TSST	Configures Test Signal Source (TSS)	38
TTTEST	Two tone test; calculates intermodulation distortion	110
GENERAL	Format conversion routines such as binary to BCD, single to double precision, fixed to floating point, floating to fixed point	500
UTILITIES**	I/O handlers and interrupt routines	500
TOTAL		4504

*Lines of code for these subroutines are included in the count for the referenced routine of which they are an entry.

**This software does not exist in the current PATE; it is peculiar to microprocessor-based system.

Table 5-7. DDMS Software

NAME	FUNCTION	LINES OF CODE
DEDEMS	Equipment Configuration, Data Collection	535
DELAY	Delays program execution for specified number of milliseconds	20
GUBOUT	Determines software timeout	50
SISP	Standard interrupt service program	110
ZAPP	Calculates DDMS parameters	117
UTILITIES	Perform arithmetic operations, number conversion, manipulate floating point numbers	591
DLC*	Data Line Controller - Receives inputs from higher level system, transmits outputs, error delection and correction, data buffering	500
GIH*	General interrupt handler	100
TOTAL		2023

*This software does not exist in the present PATE; it is peculiar to the microprocessor-based system.

Table 5-8. MSMS Software

NAME	FUNCTION	LINES OF CODE
ADRI	Performs all data input	187
DEDEMS	Collects digital P/D data, calls ZAPP to calculate parameters	493
FSKIN	Inputs data from analog P/D and A/D for an MSMS subchannel	308
GACK	Reads overflow on A/D and compares with allowable number of overflows	59
GTOR	Selects gain, channel, impedance, filter A/D rate	114
DLC*	Data line controller - receives inputs from higher level system, transmits outputs, error detection and correction, data buffering	500
GIH*	General interrupt handler	100
GUBINI	Timeout subroutine	50
LEVEL	Calculates level of composite channel	77
MSCHMO	Configures equipment for MSMS subchannel measurement, controls MSMS subtask	203
PRCAL	Calculates analog MSMS parameters	273
ZAPP	Calculates digital MSMS parameters	114
UTILITIES	Perform arithmetic operations, number conversion, manipulate floating point numbers	901
TOTAL		3379

*This software does not exist in the present PATE; it is peculiar to the microprocessor-based system.

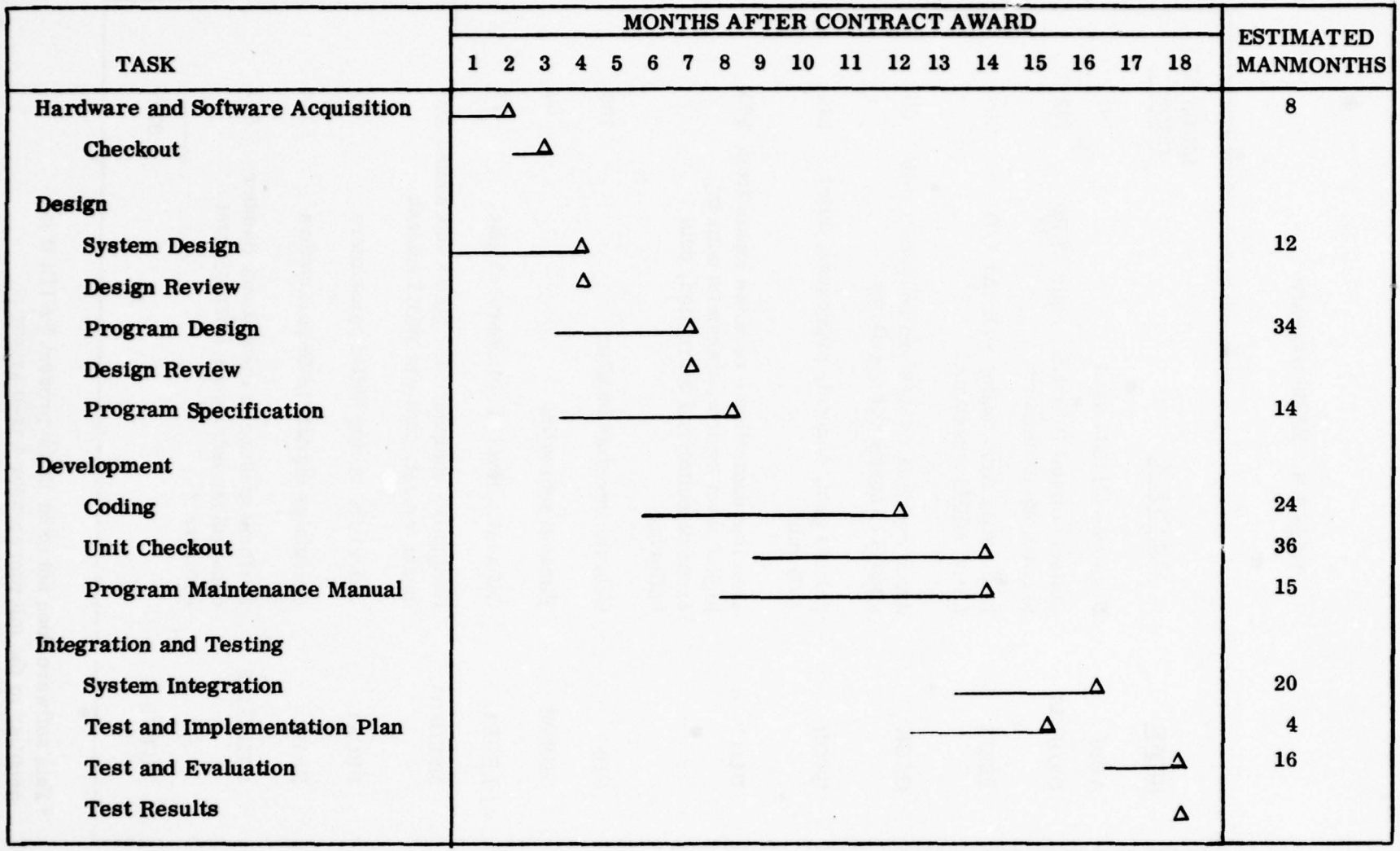


Figure 5-1. Implementation Schedule

SECTION 6 - MAINTENANCE

6.1 INTRODUCTION

This section defines the maintenance methods applicable to a microcomputer implementation of the PATE software, compares maintenance problems and methods applicable to both the present minicomputer and proposed microcomputer implementation, and presents relative advantages and disadvantages of both implementations from a maintenance point of view.

This study is undertaken with full understanding that maintenance represents a significant part of the operating costs associated with a system such as ATEC. As a system that, itself, is used as a maintenance tool, reliability is a prime ATEC requirement. Reliability and availability of a system as complex as ATEC, in turn, depend heavily on ease of maintenance, availability of spare parts, self-checking and diagnostic capability and level of training required for maintenance personnel. Thus, a PATE, as an element of ATEC, must ideally be inherently reliable, modular, and easily maintained by personnel with a low skill level. The maintenance personnel must have low cost, easy to replace spare parts constantly available. Any feasibility study of a microcomputer implementation of PATE functions must therefore carefully consider the maintenance aspects associated with such a system. This section addresses these factors and compares the maintenance requirements of such a system with the present minicomputer implementation of the PATE.

6.2 SURVEY OF COMPUTER MAINTENANCE

6.2.1 General

There are two specific differences between microcomputers and larger computers from a maintenance viewpoint. First, larger computer systems generally contain a large number and variety of peripheral devices - disks, mag tapes, line printers, teletypewriters, etc. These devices usually have many critical mechanical components, which are far less reliable than solid state electronic components and require frequent

preventive maintenance to keep them running. Microcomputers, whose uses are generally restricted to replacement of hardwired electronic circuitry and data processing functions not requiring access to large data bases, do not usually contain mechanical devices. Preventive maintenance is, therefore, less of a requirement in a microcomputer system.

The second major difference is that microcomputers can generally be serviced by non-computer-oriented personnel. This is a direct result of the extensive use of LSI devices in the microcomputer. If a microcomputer suddenly fails to execute a previously working program, it is generally not necessary to trace down the source of the problem. Such a problem is normally either an instruction failure or a memory failure. Since many microcomputers are wholly contained on a single, low cost board, any such failure can be corrected in the field by replacing the microcomputer module. In the event the microcomputer occupies more than one board, the organization will usually be such that the CPU is on one board and memory on the others. Swapping a spare module for either will, again, isolate the problem down to the board level. This is in direct contrast to larger computers where the CPU and associated timing, multiplexing, and control logic are usually spread over many boards. In this event, it is generally necessary to isolate the particular instruction or type of data transfer that fails in order to repair the computer at the board level. This requires a higher skill level than is normally found in personnel trained to service hardwired systems.

6.2.2 Hard Failures

Hard failures within a computer system are those which recur each time the computer is set to running a particular program in some particular, well defined manner. Hard failures are fairly easily isolated using standard servicing tools and techniques for digital systems (i.e., oscilloscopes, logic probes, logic scopes, etc.). Trained digital electronic technicians can usually learn to service computers with hard failures easily. Also, a hard failure can always be located by the "shotgun method", i.e., replacing modules or components, one at a time, until the problem goes away. No special diagnostics or programming knowledge are thus required to

repair computers with hard failures (although these may speed up the fault isolation process). Unfortunately, most problems in large computers are of the soft rather than the hard failure type.

6.2.3 Soft Failures

Soft failures within a computer system are those which appear only occasionally. The source may be electronic noise or a similar stochastic process; however, it is more often an obscure hard failure. In the latter case, the failure appears soft only because it is related to the placement of an instruction or data. Thus, in a computer system where programs and data are moved around in main memory, swapped into available areas of bulk storage, placed randomly with respect to other programs or data (especially in the event of pattern sensitive failure modes), or where programs are frequently interrupted by external events, hard failures of specific storage locations, pattern sensitive failures of busses or storage media, or even logical errors within a program, can appear to be soft. Based on these considerations, it is obvious that soft errors are far more likely to occur in larger computer systems than in a microcomputer. This is because larger computer systems generally strive to achieve economic feasibility by multiplexing tasks (multiprogramming), with the attendant swapping of programs and data in and out of mass storage. In such a system, a program can end up being executed out of different areas of varied bulk storage media each time it is run. In addition, previously undetected bugs in system software, whose function is to keep different programs and data areas apart in multiprogramming systems, can cause soft errors by selectively modifying the same program in many different ways, depending on the other programs being run along with it. It is obvious that simple, hardware oriented, maintenance approaches are not suitable to computer systems prone to soft failures. "Shotgunning" out circuit boards, even in a minicomputer-sized system with soft failures, is risky since one can never be sure if the problem has been solved when a board is replaced (if a problem occurs infrequently, the program must be run many times after each repair attempt to gain confidence that the problem has been corrected). Thus, correction of a soft failure usually requires

either tracing the problem right down to its source or turning it into a hard failure via creation of the right kind of free standing program. It is to this end that diagnostic programs must be provided. Diagnostics are discussed further in Paragraph 6.2.4. Computers prone to soft failures will require attention by maintenance personnel familiar with general programming techniques, the system software being used, and the use of available diagnostic programs. This is in addition to knowledge of digital circuitry, hardware debugging techniques, and service equipment. Microcomputers are generally dedicated to a single computational task, have few, if any, bulk storage peripheral devices, and are obviously far less prone to soft failures than larger computers. Thus, personnel with a much lower level of training and capability can be assigned to maintain microcomputers than to repair larger computer systems. That this conclusion can be extrapolated out to systems in which a network of microcomputers is used to replace a single, larger computer is the subject of Paragraph 6.3.

6.2.4 Role of Diagnostic Programs

It has been demonstrated that diagnostic programs are useful in any size computer system to help track down failures. For microcomputers, which are far more prone to hard failures than soft failures, and which are largely disposable (the whole microcomputer exists on one or two circuit boards, costing less than $1,000 each), the need for diagnostics is diminished. Diagnostic programs, and the attendant need for maintenance personnel who know how to run and use them, are typically required in larger more complex systems (such as the present minicomputer PATE implementation).

The goal of diagnostic programs is to isolate a problem inside a computer system. Ideally, a diagnostic package should be able to isolate any given problem automatically, with no need for creative intervention by humans. In practice, this is virtually impossible. If an instruction or memory failure is the cause of the problem, it is likely that diagnostic programs will not run correctly. If the problem is with a peripheral device, a program written to anticipate the existing failure

explicitly will probably isolate the problem. However, in the case of soft failures, prior anticipation of the failure mode is unlikely, since the number of failures that can result from the interaction of two or more basic operations in a complex system is astronomical. Therefore, well thought out diagnostic packages generally do not strive to automatically isolate soft failures. Rather, they attempt to run elements (memories and busses) within the computer very hard (i.e., with "worst case" patterns and conditions) and in combination with other critical elements to duplicate a soft failure originally detected while running operational software. This frees maintenance personnel from some of the complexity involved in real-time, multi-programming operating systems and the unfamiliar programs being executed under them. As a free standing program, a diagnostic that reproduces a failure at least assures that the problem is not in software because it is brought out again by something within that particular diagnostic program. Well designed diagnostic programs, then, have the capability of being run in pieces, so that the problem can be further isolated down to a small, easily analyzed routine. At the same time, the execution of only the section of code that causes the problem will make the failure appear more hard, hopefully allowing hard failure maintenance techniques to be used.

In some cases, diagnostic programs alone will not be sufficient to allow repair of a computer failure. This is true wherever the failure is a bug in software. In addition, unanticipated failure modes (always present with a complex machine) are not reproduced. In either event, consulting personnel with an extensive software (and sometimes engineering as well) beckground often have to be called in to help isolate the problem. The general knowledge and experience of these consultants is necessary to deduce the nature of the problem. The length of time it takes (and, hence, the cost) to isolate a soft failure of this type is variable. However, the cost of repair and system downtime in these cases is usually high.

In summary, the image of a highly trained computer maintenance man spending many hours running diagnostics and consulting with factory experts in order to repair a computer, is more common in larger computer systems than smaller. In particular, since microcomputers are generally prone to hard, rather than soft, failures (and are by and large disposable), the need for diagnostic programs and computer experts to write and use them is minimal. Microcomputer repair, in contrast to larger computer systems, involves, after receiving a report of a solid failure, only the need to find the culprit circuit board and then perform the repair using techniques and equipment familiar to hardware rather than computer repair men. This level of skill is generally compatible with graduates of military or civilian basic electronics courses plus repair school on the system to be maintained. Computer, or software, schooling for field maintenance personnel should not be necessary with microcomputer-based systems.

6.2.5 Level of Repair

Any discussion of maintenance techniques for electronic equipment must address the question: "What is the basic unit of replacement in the field?" Highly critical situations often must be handled by replacement of a whole chasses. This, however, is undesirable, due to the large expense involved and the storage and mobility of replacement parts. Much material cost reduction can be obtained by board replacement maintenance rather than chassis replacement. At this level, however, it is imperative that each unit be designed in such a manner that location of a fault to a particular board is fast and simple. If not, labor and downtime costs can easily diminish the economies gained in stocking and shipping spare boards rather than chasses. The lowest level of field maintenance is the component level. Although an electronic component is the least expensive field replaceable item in an electronic system, there is generally no way to design a circuit so that a system-level failure is quickly and easily traced to one component. Additionally, components not mounted in sockets

may require considerable time to replace (unsoldering and resoldering) which adds to labor and downtime costs; consequently, field maintenance at the component level is rarely considered. Therefore, field repairs are almost always limited to replacement of whole chassis or individual boards. Chassis replacement is characterized by high material cost and logistics problems, whereas board replacement usually involves increased downtime. It is worthy to note that a microcomputer that is wholly contained on a single board achieves the best of both approaches, i.e., it is a fully functional unit that is replaced intact by inserting a new board. No other type of computer (or even many hardwired logic circuits) can claim this advantage. Therefore a microcomputer implementation of a function can be expected to achieve an all around maintenance savings (in materials, labor and system downtime) over most other possible system implementations.

6.2.6 Preventive Maintenance

Preventive Maintenance (PM) measures, usually associated with minicomputer and larger computer systems, are undertaken for two reasons. First, computer systems generally have a number of mechanical devices associated with them (rotating components of disks, tapes, line printers, teletypewriters, and blowers) which require periodic adjustment, lubrication, cleaning and filter changes to keep them running reliably. Second, systems prone to soft failures need to be inspected regularly, since occasional problems might cause errors in system operation which are undetected by the operators. This PM function is generally implemented either by running a selected set of computer diagnostics, running special "calibration" programs (perhaps in conjunction with special test equipment), or both. While these criteria are applicable to the present PATE minicomputer implementation, they are not generally applicable to a microcomputer system. The lack of mechanical elements and the propensity toward hard rather than soft errors means that PM costs can be greatly reduced with a microprocessor system. This has the added bonus of reducing downtime and increasing system availability, important criteria in a test and maintenance system such as ATEC.

6.2.7 Microcomputer Reliability

Since microcomputers consist almost exclusively of solid state, digital electronic components, they rate among the highest reliability approaches to any problem they can solve. The lack of mechanical components often required for economic feasibility of larger computer systems, makes the microcomputer stand alone with respect to computer reliability. The microcomputer's lack of analog circuitry found both in larger computers (core memory driver/sense amplifier, magnetic storage devices, etc.) and hardwired computing circuits also sets it apart from other implementations when it comes to reliability. Additionally, the heavy use of large scale integration (LSI) technology in microcomputers is another reliability advantage, by virture of the greatly reduced parts count over other approaches. Thus, a microcomputer-based system can be expected to be superior to other system realizations in terms of reliability. This, of course, results in across the board maintenance savings in materials, labor and downtime.

6.3 MICROCOMPUTER MAINTENANCE

6.3.1 General

Paragraph 6.2 discussed the computer maintenance problem in general and concluded that microcomputers do not share most of the problems or cost associated with maintaining larger computers in the field. However, since a microcomputer is a lower capability device than larger computers, it generally takes a network of several microcomputers to replace one minicomputer (or larger computer) in any given system. It is natural, therefore, to ask whether this multiplicity of microcomputers will impact on the advantages previously demonstrated for a single microcomputer. This question is addressed in the paragraphs that follow. The approach used is to present a feasible microcomputer implementation of a PATE function as a baseline for discussion and analysis. This baseline system is presented in Paragraph 6.3.2.

6.3.2 Microcomputer Implementation of a PATE Function

Feasible implementation of the PATE IQCS function using microcomputers is shown in Figure 6-1. A more detailed discussion of this system is presented in Section 3. For the purposes of discussing maintenance, it is assumed that the system of Figure 6-1 will work, and will be used as a baseline for further discussion of maintenance.

An important aspect of the system shown in Figure 6-1 is that it is broken down along functional lines, i.e., each microcomputer performs one or more complete functions. Thus, the system, as shown, more closely approximates a hardwired implementation of an IQCS function than a minicomputer-based PATE. This is extremely important when considering maintenance, because it supports the conclusion that microcomputer-based systems can be configured to look, to maintenance personnel, like hardwired systems (only with far less circuitry). Consequently, maintenance personnel familiar with hardwired systems, but with no computer training, should be capable of maintaining the system in Figure 6-1. This is significant because military and civilian schools produce an abundance of such personnel, while maintenance personnel with computer training and experience are more expensive and generally less available.

Microprocessor A, in Figure 6-1, controls the scanners and A/D converters, acquires A/D data, and performs squaring, summing, averaging, and peak detection on this data. Microprocessor B accepts the data from microprocessor A and performs averaging, a 128 point Fast Fourier transform and subsequent processing on this data. Microprocessor C directs and coordinates the actions of all other microprocessors in accordance with instructions from a larger higher level computer. Microprocessor E manages a local I/O terminal which may be optionally included to provide local operation as a backup to a failure at the higher level. Microprocessor D manages the modem over which all communications with the larger higher level computer take place. While the services of a larger computer are retained with this implementation, only one such computer is used to handle a network of microcomputers.

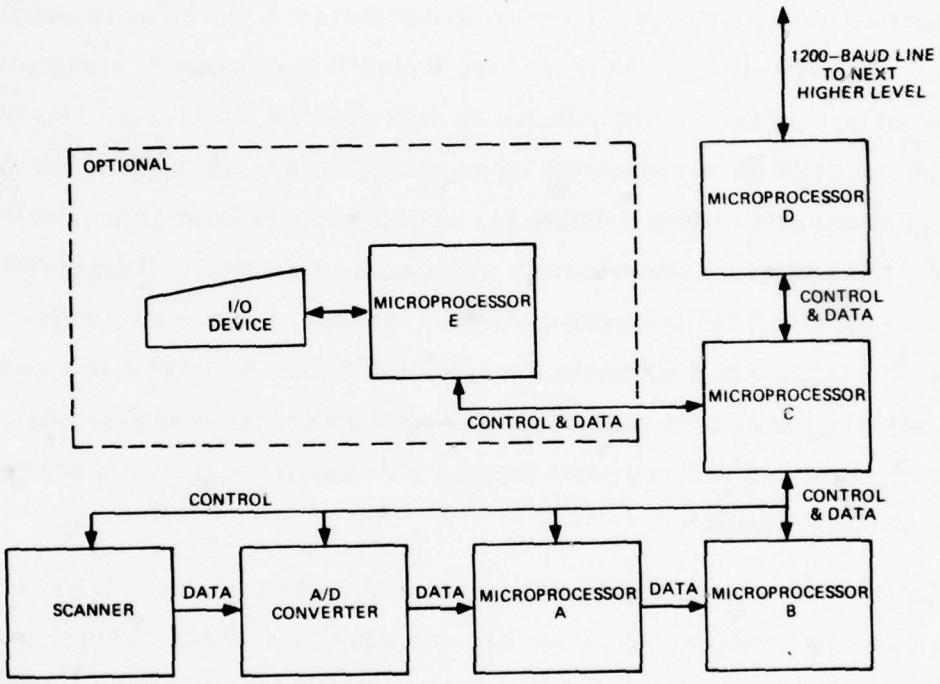


Figure 6-1. Feasible IQCS Equipment Configuration, Data Collection, and Parameter Calculation Microprocessor System

This is in contrast to the present PATE implementation which requires a minicomputer system be local to each PATE.

6.3.3 Identification of Failures

Initial indications of a problem with a system, as implemented in Figure 6-1, will probably show up at the higher level computer as an error indicator or at an intermediate level of the microcomputer chain as an overall lack of data. To assist maintenance personnel in fault isolation, each microcomputer should be provided with programs to check the validity (via parity, checksums, etc.) of its input data and to flag both local site personnel and upstream processors of any problem. In this manner, each microcomputer will have the capability of checking the previous (downstream) microcomputer and reporting failures in it. If any microcomputer reports a problem with its inputs, the fault must either be in the most downstream microcomputer reporting the problem or in the microcomputer immediately preceding it. Maintenance personnel can easily locate the source of the problem by examining the input and output control and data information to each of these microcomputers. Since the system is similar to a hardwired configuration (i.e., each element handling one or a few distinct calculations in a chain), repair personnel familiar with hardwired systems (and with the PATE system functions) should have little trouble locating the source of the problem down to the chassis (microcomputer) level. Since each microcomputer will consist of only a few (generally one or two) boards, repairs can be made at the board level by "shotgunning" in spare cards. Consequently, one design goal of a microcomputer-based system should be to make the system appear to maintenance people as hardwired. All that is necessary to achieve this, in the case of the PATE IQCS function, is to incorporate a few data checking routines into the microcomputer along with some programs and interfaces to drive familiar looking status displays (alarm lamps, meters, digital readouts etc.) which will be recognizable to people familiar with PATE system functions.

One consequence of making a microcomputer-based system appear hardwired is that the microcomputer software components will have to be placed in read only

memory (ROM). This will prevent accidental alteration of programs during operation and maintenance, an important factor for microcomputers where no backup load capability (i.e. mass storage) is available. A consequence of the placement of software in ROM is that software errors found in the field cannot be corrected there. New ROMs will have to be prepared at the system development site and sent out to the field for replacement. This is not seen as a obstacle, however, because it is unlikely that problems will be traced to software in the field. This is particularly true where field maintenance personnel do not have a software background or training.

If a software "bug" does slip through test and evaluation and into production units, maintenance people in the field will note that replacing all circuits did not cure the problem. Engineers, programmers, and analysts involved in system development will then be called into the system development site where the problem will be simulated (based on field reports of the conditions under which the problem arose). It is these people who will ultimately find the "bug" and work out the fix for it. Hence, no problem will arise if they prepare new ROMs with repaired and tested software and send these new memories out to the sites for installation by field maintenance personnel. The updated system may then be retested in the field on a functional basis to ensure that both the software fix and the hardware installation are correct.

The software system maintenance function would be performed at an ATEC system development site. This site would most likely be located in CONUS and contain a complete microprocessor development system and be staffed with hardware specialists, software system analysts, programmers and SYSCON operation specialists.

In summary, since microcomputers are small and inexpensive, they may be configured to resemble a hardwired system implementation. This should certainly be done with ATEC, because of the easy availability of people trained in the repair of such systems and the attendant reduction in maintenance costs. Identification of a system problem can thus be performed in a manner identical to that used for hardwired systems. When a fault is isolated to a particular unit (chassis), however, repair is even easier than with hardwired systems since microcomputers, making

extensive use of LSI technology, require far fewer circuit boards to implement any particular function. Thus, a simple replacement approach can be used in the field to isolate the faulty circuit board.

6.3.4 Logistics and Level of Repair

From a cost viewpoint, field maintenance is optimal at the board level, although there are exceptions. However, stocking spare boards is far less costly than spare chasses and it generally does not take much more time for a trained repairman to isolate a faulty board after the malfunctioning chassis has been found. This is particularly true where circuit boards are functional, i.e., each board implements a complete function on its input signals. The major problem with functionally organized boards in a hardwired environment is that many different types of boards exist within any one system. This tends to increase production and inventory costs. However, a microcomputer-based system does not generally suffer from this problem. A microcomputer, when used to implement any one of a large number of functions, may appear externally as if it were hardwired. Internally, however, the microcomputer is a general purpose hardware unit. It is programmable; the same hardware can be configured to perform different tasks. The mechanism for performing this configuration in a microcomputer is software - usually stored in an ROM. Therefore, a single type of board (perhaps two or three where the whole microcomputer cannot be contained on one board) can be inventoried - along with ROMs containing programs for all required functions. The costs and logistics problems associated with sparing modules in a microcomputer-based system are therefore much less than in hardwired systems. The general purpose nature of microcomputers is enhanced by the recent trend toward incorporating programmable I/O circuits into microcomputers. Modern microcomputers on a board are not only programmable from the standpoint of the calculations and logic that they perform, but also in the nature of their interfaces with the outside world (including one another). Hence, modern microcomputers make the objective of a truly universal circuit module a practical reality.

Repair of electronic equipment at the component level is rarely considered for the field due to the excessive time it can take to locate a faulty component on a circuit card using general purpose, field oriented, test equipment. Special purpose test equipment, geared to isolating faults on a particular functional module can greatly speed up this process. However, such equipment, being dedicated to a particular type of circuit as it is, is generally very expensive and not sufficiently portable or rugged for use in the field. Thus, repair of faulty boards from the field is usually performed (if at all) at a central depot equipped with the necessary special purpose test gear and trained personnel. This solution is probably the best for microcomputer modules as well, since repair at the chip level usually requires running diagnostic programs on a specialized test stand designed specifically to "debug" a particular microcomputer. Of course, a microcomputer need not be repaired at the component level at all. Most microcomputer boards suitable for use in PATE currently cost less than $1,000, and prices are decreasing. Thus, a microcomputer module will soon be no more expensive than a hardwired circuit board. The same criteria can be applied to component level repair of a microcomputer as to any other type of circuit card. The cost considerations may well be such that a microcomputer within ATEC can eventually be treated as disposable.

An assumption has been made that all the microcomputers shown in the baseline system (Figure 6-1) are the same type. This assumption is very important. From the point of view of a systems engineer, some baseline system functions might be better accommodated by one microcomputer while other functions will favor another. For example, a character-oriented function whose requirements are more in the nature of control than calculation is generally considered most compatible with an 8-bit MOS microcomputer (such as the M6800). A high speed, high resolution "number crunching" task, on the other hand, might require a 16-bit bipolar unit. From a purely technical standpoint, different microcomputers should be considered for these tasks. However, when these two functions are part of the same system, using different microcomputers will result in increased production, logistics, and maintenance costs. Thus, given a similar case, careful consideration should be

given to the feasibility of accommodating both functions with a single type of microcomputer. Perhaps this system could be repartitioned to allow a compromise, 16-bit MOS microcomputer to be the only type used. In any event, savings realized by implementing a simple function with a simple microcomputer may be more imagined than real. The cost of even a very powerful and fast microcomputer is still low compared to inventory and labor costs. Selection of a single microcomputer, suitable for most, if not all, of the PATE functions should therefore be a prime consideration in the system design.

6.4 RESULTS AND CONCLUSIONS

Based on the preceding discussions, the following conclusions can be drawn concerning maintenance problems in a microcomputer-based system versus the present minicomputer implementation:

1. Microcomputers are more reliable than minicomputers. They are generally all solid state units and have far fewer components than minicomputers.

2. Microcomputers are less costly to maintain. They are board level devices, may be considered disposable, and reduce inventory costs because they are programmable logic.

3. Microcomputer-based systems can be simpler to service than minicomputer systems. A functionally designed network of microcomputers can be made to look like hardwired logic, allowing the use of maintenance personnel who do not understand computers or software to isolate and repair failures.

In conclusion, a microcomputer implementation of PATE functions is highly desirable from the maintenance aspect. It has many advantages over the present minicomputer approach, primarily in added reliability, reduced downtime, and cost, while suffering no major disadvantage.

SECTION 7 - CONCLUSIONS

This section presents the conclusions of the ATEC Software Conversion Study effort. It identifies the software recommended for conversion to microprocessor implementation and summarizes the results of the study.

7.1 MICROPROCESSOR-BASED SOFTWARE

The existing PATE software, as defined in Reference 1, was examined in detail to determine the feasibility of implementing all, or portions, of it on microprocessors. This software consisted of approximately 67,000 lines of Honeywell H316 minicomputer Assembly Language code. Results of this examination indicated that portions of the four measurement tasks (IQCS, OQCS, DDMS, and MSMS) were amenable to microprocessor implementation. Each of the four measurement tasks consist of a set of software modules which are functionally oriented. The tasks were subjected to a detailed analysis at the module level to determine which were suitable for microprocessor implementation. The main factors considered were:

- Data base interaction
- Operator and operating system interaction
- Storage requirements
- Impact on throughput
- Opportunity for parallel processing.

The resulting division of software in this study between that which can be converted to microprocessors and the remaining software which would be implemented in the higher level system is a function of the conversion criteria used and the design of the present PATE software and data base. The software recommended for conversion to microprocessors in this study includes portions of the IQCS, OQCS, DDMS and MSMS measurement tasks. The higher level system would perform the remaining PATE functions and would aggregate common functions currently performed by multiple PATE's such as alarm analysis.

The functions performed by the software recommended for conversion to microprocessors all fall within the Measurement Acquisition Subsystem (MAS) Level V of the five level System Control hierarchy. These microprocessors, however, would not perform all of the functions required of the MAS, since many of the functions examined under the PATE architecture, such as alarming and operator interface, were recommended for implementation at the higher level system. Thus, the higher level system would include some of the functions required of both the Station Control and Nodal Control functional areas. Obviously, if other software and data base architectures were considered, additional functional requirements could possibly be implemented at the MAS by microprocessors. This consideration was not part of this study effort however, but should be pursued. The study does demonstrate, however, that for the PATE software considered, a significant portion of the MAS function could be implemented by microprocessors with a minicomputer performing the remaining functions including the man-machine interface for the Station Control Subsystem.

The following paragraphs summarize the results of this analysis.

7.1.1 *IQCS Measurement Task*

It was determined that the IQCS Equipment Configuration, Data Collection, and Parameter Calculations module was implementable on microprocessors. The IQCS Task Control and Task Termination modules were assigned to the higher level system because of their significant operating system interaction. The Alarm Analysis, Trend Control, and Output Formatter modules were also put at the higher level due to their need for frequent data base access, complex calculations, and operator interaction.

The feasibility of a microprocessor implementation of the IQCS Equipment Configuration, Data Collection, and Parameter Calculations module was demonstrated using a configuration containing five microprocessors, one of which is optional. All microprocessors in the system have a 16-bit word and hardware multiply and divide. System memory requirements are approximately 6K words. The data base is maintained at a higher level and is accessed by the microprocessor-based system over a communications line.

7.1.2 OQCS Measurement Task

The OQCS Test Controller module was found to be amenable to microprocessor implementation. The OQCS Final Calculations, Alarm, and Output Formatter modules were assigned to a higher level due to the complexity of the calculations, a high degree of data base interaction, and the need for operator interaction.

The OQCS Test Controller module was implemented as an additional plug-in microprocessor in the IQCS system, thereby creating a combined I/OQCS microprocessor-level system. An additional 5K words of memory are required. The microprocessor characteristics and communications requirements were similar to those described for the IQCS.

7.1.3 DDMS and MSMS Measurement Tasks

In both the DDMS and MSMS Measurement Tasks it was determined that the Equipment Configuration, Data Collection, and Parameter Calculations module could be implemented on a microprocessor. The Task Control and Task Close Out modules were assigned to the higher level system because of significant operating system interaction. The Alarm and Tend Control modules were assigned to the higher level because of extensive data base access requirements and the Output Formatter modules were similarly assigned to the higher level system because of the required data base access and operator interaction. The MSTM subroutine of the MSMS Equipment Configuration, Data Collection and Parameter Calculations module was also assigned to the higher level system because of extensive data base interaction requirements and operator interaction.

Microprocessor implementation of each of the Equipment Configuration, Data Collection, and Parameter Calculations subtasks requires one 16-bit microprocessor with hardware multiply and divide, and 8K words of memory to contain programs, data, and buffers. The data base would be stored in a higher level computer system, and any data base items required by the microprocessor would be transmitted over communications lines. Similarly, outputs from the microprocessor would be transmitted over a communications line to the higher level computer system.

7.1.4 Microprocessor Software Implementation

It was determined that development of the microprocessor software for the recommended modules of the I/OQCS, DDMS, and MSMS Measurement Tasks would require approximately 9000, 2050 and 3400 lines of code, respectively. This development effort would cost approximately $732,000 and require 18 months.

7.2 MICROPROCESSOR FEASIBILITY DEMONSTRATION

The feasibility of implementing the DDMS Equipment Configuration and Data Collection functions was demonstrated by a sample conversion of those functions to the Motorola M6800 Assembly Language and their simulation using a Microtec M6800 software simulator. It was found that this microprocessor implementation requires approximately the same amount of storage as the Honeywell H316 minicomputer, and takes about twice as long to execute a program with a small amount of arithmetic operations.

REFERENCES

1. Prime Item Development Specification for Programmable ATEC Terminal Element (PATE) of Communications Performance Monitoring and Assessment System AN/GYM-12(V)(XW), Parts I and II, February 1976 and June 1976, Specification No. CP75000872 Rome Air Development Center, Griffiss Air Force Base, Rome, New York 13441.

2. 990 Computer Family Systems Handbook, Texas Instruments Inc., Manual Number 945250-9701, 2nd Edition, December 1975.

3. Series 16 Models 316 and 516 Programmer's Manual, Honeywell Information Systems Inc., Document Number 70130072156E, November 1976.

4. Addendum to Prime Item Development Specification for In-Service/Out-of-Service Quality Control Subsystem with Modern Signal Monitor Subsystem Option and Caelus Disk of Communications Performance Monitoring and Assessment System AN/GYM-12(V) (XW), Part I of II Parts, 16 October 1974. Specification Number 75000869D, Rome Air Development Center, Griffiss Air Force Base, Rome, New York 13441.

APPENDIX A - H316/MICROPROCESSOR COMPARISON

A.1 PURPOSE

A study was undertaken to investigate the performance of current 8-bit and 16-bit microprocessor chips and compare them with the H316 minicomputer. Three microprocessors were chosen from those currently available. They were:

- Motorola M6800 (8-bit)
- General Instruments Series 1600 (16-bit)
- Texas Instruments TMS9900 (16-bit).

A.2 METHODOLOGY

A subroutine was selected from among those available in the IQCS portion of the H316 PATE software. Particular attention was given to the instruction mix when selecting the subroutine, as it was the intent of this study to compare an application which contained a mix of the most heavily used instructions and a representative amount of arithmetic operations. The subroutine selected was the FFT algorithm. In addition to its acceptable instruction mix, it has no external subroutine calls which impact on the mix and all arithmetic is single precision integer (16 bits).

Microprocessor and the Honeywell H316 performance characteristics were determined in the following manner:

- H316 - instruction execution times were taken from the Models 316 and 516 Programmer's Reference Manual
- M6800 - instruction execution times were derived using information contained in the M6800 Programming Manual
- Series 1600 - instruction execution times were derived using information contained in the Series 1600 Microprocessor System Semiconductor Documentation Manual

- TSM9900 - instruction execution times were derived using information contained in the Texas Instruments 990 Computer Family Systems Handbook.

Table A-1 shows a comparison of the instruction execution times for the four processors; Table A-2 provides an instruction mix versus available instruction execution time tabulation; Table A-3 contains total execution time for each instruction computed as the product of its execution time and associated mix percentage.

A.3 CONCLUSIONS

As can be seen in Table A-2, the fact that the M6800 and Series 1600 do not contain a multiply in their instruction repertoire has a significant impact on the distribution of time available to the machine for each operation. The load, store, and branch operations constitute over 67 percent of the mix but take slightly over 30 percent of the total time on the M6800 and Series 1600 as opposed to an average of 56 percent of the total time on the H316 and the TMS9900. Algorithms requiring multiplication or division in as little as 5 percent of the total instruction mix will severely tax the computing power of either the M6800 or Series 1600.

Another factor to be considered is the 8-bit word size of the M6800. Although its instruction execution time is slightly faster than that of the Series 1600 or the TMS9900 it has no double precision (16-bit) arithmetic instructions in its repertoire; therefore, conversion of arithmetically oriented routines from the H316 will be quite expensive in terms of both program size and execution time when compared to conversion to 16-bit micros with comparable instruction times.

As can be seen from the Table A-3 totals, the computed difference in execution times between the H316 and either the M6800 or Series 1600 is approximately 4 to 1 when the times dedicated to the software multiply routines of the microprocessors are included. If the multiply times are disregarded, the ratio decreases to 2 to 1 which is competitive with the TMS9900.

Table A-1. Instruction Execution Times

OPERATION	HONEYWELL H316	MOTOROLA M6800	GENERAL INSTRUMENTS SERIES 1600	TEXAS INSTRUMENTS TMS9900
	INSTRUCTION EXECUTION TIMES (microseconds)			
Load	3.2	5.4	6.8	6.7
Store	3.2	6.4	6.4	6.7
Conditional Branch	1.6	5.2	4.0	5.0
Unconditional Branch	1.6	5.2	4.0	5.0
Add	3.2	5.4	6.4	7.3
Subtract	3.2	5.4	6.4	7.3
Multiply*	8.8	125.0	130.0	20.0
Shift	2.4	3.2	3.2	11.3
Other**	3.0	5.2	5.2	9.8

*The H316 and TMS9900 multiply operations are hardware functions; the M6800 and Series 1600 do not have a multiply instruction. The times shown are for execution of an unsigned multiply subroutine giving a 16-bit result.

**The time for "other" operations was computed as an average of the times given for each instruction in the machine's repertoire.

Table A-2. Instruction Mix Versus Execution Time

OPERATION	PERCENT OF MIX	HONEYWELL H316	MOTOROLA M6800	GENERAL INSTRUMENTS SERIES 1600	TEXAS INSTRUMENTS TMS9900
		PERCENT OF EXECUTION TIME			
Load	19.81	20.87	9.07	10.40	17.44
Store	21.74	22.90	11.08	12.13	19.14
Conditional Branch	13.04	6.87	5.75	4.28	8.62
Unconditional Branch	13.04	6.87	5.75	4.28	8.62
Add	5.80	6.11	2.16	3.04	6.25
Subtract	4.35	4.58	1.99	2.28	4.20
Multiply*	5.31	15.39	56.34	56.68	14.04
Shift	1.93	1.53	.52	.51	2.89
Other	14.98	14.79	6.60	6.39	19.45

*The H316 and TMS9900 multiply operations are hardware functions; the M6800 and Series 1600 do not have a multiply instruction. The percentages shown are for execution of an unsigned multiply subroutine giving a 16-bit result.

Table A-3. Total Execution Time By Operation

OPERATION	PERCENT OF MIX	HONEYWELL H316	MOTOROLA M6800	GENERAL INSTRUMENTS SERIES 1600	TEXAS INSTRUMENTS TMS9900
		TOTAL EXECUTION TIME (microseconds)			
Load	19.81	63.39	106.97	134.71	131.93
Store	21.74	69.57	139.14	139.16	144.79
Conditional Branch	13.04	20.86	67.81	52.16	65.20
Unconditional Branch	13.04	20.86	67.81	52.16	65.20
Add	5.80	18.56	31.32	37.12	42.34
Subtract	4.35	13.92	23.49	27.84	31.76
Multiply*	5.31	46.73	663.75	690.30	106.21
Shift	1.93	4.63	6.18	6.18	21.85
Other	14.98	44.94	77.90	77.90	147.10
TOTAL	100.00	303.46	1184.36	1217.53	756.37
RELATIVE SPEED			3.9 to 1	4.0 to 1	2.5 to 1

*The H316 and TMS9900 multiply operations are hardware functions; the M6800 and Series 1600 do not have a multiply instruction. The times shown are for execution of an unsigned multiply subroutine giving a 16-bit result.

APPENDIX B - MOTOROLA M6800 MICROPROCESSOR

- NMOS TECHNOLOGY
- EIGHT BIT WORD
- EIGHT BIT DATA BUS
- SIXTEEN BIT ADDRESS BUS
- 72 INSTRUCTIONS
- SEVEN ADDRESSING MODES
- ADDRESSABLE STACK
- INTERRUPTS (maskable and non-maskable)
- SIX REGISTERS:
  - TWO ACCUMULATORS
  - INDEX REGISTER
  - PROGRAM COUNTER
  - STACK POINTER
  - CONDITION CODE REGISTER

APPENDIX C - MICROPROCESSOR CHARACTERISTICS

TEXAS INSTRUMENTS TMS9900

- NMOS TECHNOLOGY
- SIXTEEN-BIT DATA WORD
- SIXTEEN-BIT ADDRESS BUS
- 69 INSTRUCTIONS
- HARDWARE MULTIPLY AND DIVIDE
- SIXTEEN USER DEFINABLE OPERATION CODES FOR IMPLEMENTING SPECIAL PURPOSE ALGORITHMS USING EITHER HARDWARE OR SOFTWARE
- FIVE ADDRESSING MODES
- EIGHT VECTORED INTERRUPTS
- REGISTERS:
  - PROGRAM COUNTER
  - STATUS
  - WORKSPACE POINTER
  - GENERAL REGISTERS ARE ASSIGNED AS GROUPS OF SIXTEEN MEMORY LOCATIONS USING THE WORKSPACE POINTER

DATA GENERAL MICRONOVA

- NMOS TECHNOLOGY
- 16-BIT DATA WORD
- 15-BIT ADDRESS BUS
- 202 INSTRUCTIONS
- HARDWARE MULTIPLY/DIVIDE
- HARDWARE STACKS
- SIX ADDRESSING MODES
- VECTORED INTERRUPTS
- FOUR GENERAL REGISTERS

DIGITAL EQUIPMENT CORPORATION LSI-11

- NMOS TECHNOLOGY
- 16-BIT DATA WORD
- 16-BIT ADDRESS BUS
- 122 INSTRUCTIONS (112 standard; 10 optional)
- HARDWARE MULTIPLY/DIVIDE (optional)
- ADDRESSABLE STACKS
- SEVEN ADDRESSING MODES
- VECTORED PRIORITY INTERRUPTS
- *EIGHT GENERAL REGISTERS (including stack pointer and* program counter)

NATIONAL SEMICONDUCTOR IMP-16C/P

- PMOS TECHNOLOGY
- 16-BIT DATA WORD
- 16-BIT ADDRESS BUS
- 60 INSTRUCTIONS (43 standard; 17 optional)
- HARDWARE MULTIPLY/DIVIDE (optional)
- 16-WORD ADDRESSABLE STACK
- FIVE ADDRESSING MODES
- TWO INTERRUPTS (one general; one vectored)
- FOUR GENERAL REGISTERS

GENERAL AUTOMATION GA 16/110

- NMOS TECHNOLOGY
- 16-BIT DATA WORD
- 16-BIT ADDRESS BUS
- 91 INSTRUCTIONS
- HARDWARE MULTIPLY/DIVIDE
- 11 ADDRESSING MODES
- VECTORED PRIORITY INTERRUPTS
- 16 GENERAL REGISTERS

APPENDIX D - GLOSSARY

This glossary contains definitions of terms, abbreviations, and acronyms used in the ATEC Software Conversion Study.

TERM	DEFINITION
1A	1 kHz (no FFT)
1B	1 kHz (FFT only)
1C	1 kHz (Hit Analysis)
26	2600 Hz Test
2T	Two Tone (700Hz, 1000Hz)
ADDRESSING MODES	An address is a coded instruction designating the location of data or program segments in storage. The address may refer to storage in registers or memories or both. The address code itself may be stored so that a location may contain the address of data rather than the data itself. Addressing modes vary considerably because of efforts to reduce program execution time.
ASSEMBLER PROGRAM	The Assembler Program translates man readable source statements (mnemonics) into machine understandable object code.
ASSEMBLY LANGUAGE	A machine oriented language. Normally the program is written as a series of source statements using mnemonic symbols that suggest the definition of the instruction and is then translated into machine language.
A/D	Analog to Digital
ADIOS	ATEC Disk Operating System
ATEC	Automated Technical Control
BD	Bias Distortion
BAUD RATE	A measure of data flow. The number of signal elements per second based on the duration of the shortest element. When each element carries one bit, the baud rate is numerically equal to bits per second (bps).
BCD (Binary Coded Decimal)	Each decimal digit is binary coded into 4-bit words. The decimal number 11 would become 0001 0001 in BCD.

BYTE	Indicates a pre-determined number of consecutive bits treated as an entity. For example, 4-bit or 8-bit bytes. "Word" and "Byte" are used interchangeably.
COMPILERS	Compilers translate higher-level languages into machine code.
CPU (Central Processing Unit)	The heart of any computer system. Basically the CPU is made up of storage elements called registers, computational circuits in the ALU, the Control Block, and I/O. The one-chip microprocessors have limited storage space, so memory is added in modular fashion. Most current microprocessors consist of a set of chips, one or two of which form the CPU.
CROSS-ASSEMBLER	When the program is assembled by the same microprocessor that it will run on, the program that performs the assembly is referred to simply as an assembler. If the program is assembled by some other processor, the process is referred to as cross-assembly. Occasionally the phrase "native assembler" will be used to distinguish it from a cross-assembler.
CCSD	Command Communication Service Designator
CN	C-MSG Weighted Noise
CPMAS	Communication Performance Monitoring and Assessment System
DATA BUS	The microprocessor communicates internally and externally by means of the data bus. It is bidirectional and can transfer data to and from the CPU, memory storage, and peripheral devices.
DIRECT ADDRESSING	This is the standard addressing mode. It is characterized by an ability to reach any point in main storage directly.
DMA (Direct Memory Access)	A method of gaining direct access to main storage to achieve data transfer without involving the CPU.
DDMS	Digital Distortion Monitor Subsystem
DI	Dead Idle Test
DMC	Direct Memory Channel
DO	Dropouts
ED	Envelope Dealy

EXECUTION TIME	Usually expressed in clock cycles necessary to carry out an instruction. Since the clock frequency is known, the actual time can be calculated.
FIRMWARE	Software instructions which have been permanently frozen into a ROM are sometimes referred to as firmware.
FD	Fortuitous Distortion
FFT	Fast Fourier Transforms
FM	Stepped Fast, Modulated
FO	Frequency Offset
FR	Frequency Response
FSK	Frequency Shift Keyed
HD	Harmonic Distortion
HARDWARE	The individual components of a circuit, both passive and active, have long been characterized as hardware. Today, any piece of data processing equipment is informally called hardware.
HARD-WIRED LOGIC	Random Logic design solutions require interconnection of numerous integrated circuits representing the logic elements. An example of hard-wired logic is the use of a hard-wired diode matrix instead of a ROM. These interconnections, whether done with soldering iron or by printed circuit board, are referred to as hard-wired logic in contrast to the software solutions achieved by a programmed ROM or microprocessor.
HIGH LEVEL LANGUAGE	This is a problem-oriented programming language as distinguished from a machine-oriented programming language. The former's instruction approach is closer to the needs of the problems to be handled than the language of the machine on which they are to be implemented.
I/O	Input/Output
I/OQCS	In-Service/Out-of-Service Quality Control Subsystem
IM	Intermodulation Distortion
IQCS	In-Service Quality Control Subsystem
ISD	In-Service Digital
ISFSK	In-Service FSK
ISVF	In-Service VF
LSI	Large Scale Integration

MACHINE LANGUAGE	The only language the microprocessor can understand is binary. All other programming languages must be translated into binary code before entering the processor and decoded back into the original language after leaving it.
MAS	Measurement Aquisition Subsystem
MEMORY	The part of a computer system into which information can be inserted and held for future use. Storage and memory are interchangeable expressions. Memories accept and hold binary numbers only. Memory types are core, disk, drum, and semiconductor.
MICROPROGRAM	This word pre-dates the microprocessor and refers to computer instructions which do not reference the main memory storage. It is a computer technique which performs subroutines by manipulating the basic computer hardware and is often referred to as "computer within computer." The word has not changed its basic meaning when used in connection with microprocessors, but is not to be construed as native to microprocessors. A series of instructions stored in a ROM, any portion of which can implement a higher language program, is labeled a microprogram.
MICROPROCESSOR	The microprocessor is a Central Processing Unit fabricated on one or two chips. While no standard design is visible in existing units, a number of well-delineated areas are present in all of them: Arithmetic & Logic Unit, Control Block, and Register Array. When joined to a memory storage system, the resulting combination is referred to in today's usage as a microprocessor.
MM	Manual Modulated
MU	Manual Unmodulated
N	Normal
NC	Noise Coloration
ND	Noise Difference (C-MSG 3 kHz)
NF	Noise Frequency
OBJECT PROGRAM	The end result of the source language program after it has been translated into machine language.
OQCS	Out-of-Service Quality Control Subsystem
OSVF	Out-of-Service VF

P/D	Period to Digital
PATE	Programmable ATEC Terminal Element
PH	Phase Hits
PJ	Phase Jitter
PORT	Device terminals which provide electrical access to a system or circuit. The point at which the I/O is in contact with the outside world.
PROGRAM	A procedure for solving a problem and frequently referred to as software.
PROGRAM COUNTER	One of the registers in the CPU which holds addresses necessary to step the machine through the program. During interrupts, the program counter saves the address of the instruction. Branching also requires loading of the return address in the program counter.
PROM	Programmable read only memory
RAM	Random Access Memory
REAL TIME OPERATION	Data processing technique used to allow the machine to utilize information as it becomes available, as opposed to batch processing at a time unrelated to the time the information was generated.
REGISTER	A register is a memory on a smaller scale. The words stored therein may involve arithmetical, logical, or transferral operations. Storage in registers may be temporary, but even more important is their accessibility by the CPU. The number of registers in a microprocessor is considered one of the most important features of its architecture.
ROM	Read Only Memory
SOFTWARE	The language used by a programmer to communicate with the computer. Since the only language spoken by a computer is mathematical, the programmer must convert his verbal instructions into numbers.
SIMULATOR	A special program that simulates the logical operation of the microprocessor. It is designed to execute object programs generated by a cross-assembler on a machine other than the one being worked on and is useful for checking and debugging programs prior to committing them to ROM firmware.

SLICE	A type of chip architecture which permits the cascading or stacking of devices to increase word length.
STACK	The stack is a block of successive memory locations which is accessible from one end on a last-in first-out basis (LIFO). The stack is co-ordinated with the stack pointer which keeps track of storage and retrieval of each byte of information in the stack. A stack may be any block of successive information locations in the read/write memory.
STACK POINTER	The stack pointer is co-ordinated with the storing and retrieval of information in the stack. The stack pointer is decremented by one immediately following the storage in the stack of each byte of information. Conversely, the stack pointer is incremented by one immediately before retrieving each byte of information from the stack. The stack pointer may be manipulated or transferring its contents to the index register or vice versa.
STORAGE	The word storage is used interchangeably with memory.
SM	Stepped Slow, Modulated
SN	Signal Plus Noise/Noise
SPC	Signal Parameter Converter
SR	Spectrum Ratio Table
ST	Self Test
SYSCON	System Control
TA	Test Alignment
TTT	Test Tone Transmitter
TTY	Teletypewirter
THROUGHPUT	The speed with which problems or segments of problems are performed is called throughput. Defined in this way, it is obvious that throughput will vary from application to application. As an index of speed, throughput is meaningful only in terms of a particular application.
VF	Voice Frequency