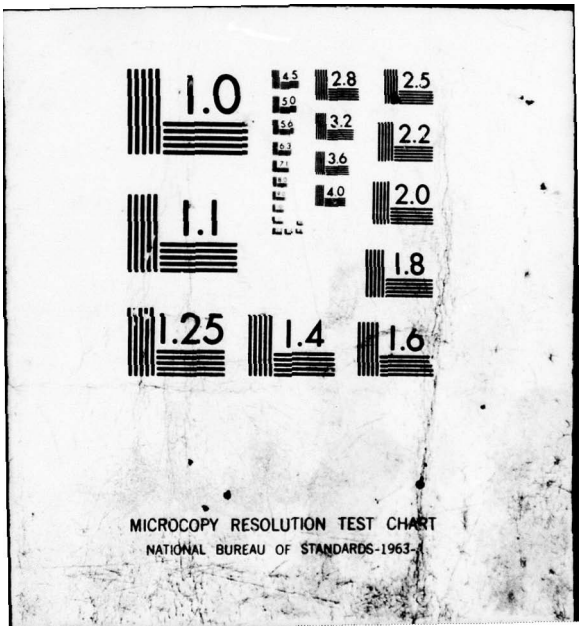
AD-A073 068 BOEING AEROSPACE CO SEATTLE WA BOEING MILITARY AIRPL--ETC F/G 17/7
EVALUATION OF DAIS TECHNOLOGY APPLIED TO THE INTEGRATED NAVIGAT--ETC(U)
MAY 79 D DEWEY, R BOUSLEY, S BEHNEN, J MASON F33615-77-C-1233

UNCLASSIFIED AFAL-TR-79-1061 NL

1 OF 2
AD A073068


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFAL-TR-79-1061

LEVEL II

EVALUATION OF DAIS TECHNOLOGY APPLIED TO THE INTEGRATED NAVIGATION SYSTEM OF A TACTICAL TRANSPORT

BOEING AEROSPACE COMPANY
SEATTLE, WASHINGTON 98124

MAY 1979

FINAL REPORT FOR PERIOD SEPTEMBER 1977 – MAY 1979

Approved for public release; distribution unlimited

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

James M Bain

JAMES M. BAIN
Project Engineer

Terrance A. Brim

TERRANCE A. BRIM
Acting Chief
DAIS Program Branch

FOR THE COMMANDER

Raymond E Siferd

RAYMOND E. SIFERD, Col, USAF
Chief, System Avionics Division
Air Force Avionics Laboratory

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFAL/AAS ,W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
(18) 1. REPORT NUMBER AFAL-TR-79-1061	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
(6) 4. TITLE (and Subtitle) Evaluation of DAIS Technology Applied to the Integrated Navigation System of a Tactical Transport.		(9) 5. TYPE OF REPORT & PERIOD COVERED Final Report. September 1977 - May 79 6. PERFORMING ORG. REPORT NUMBER
(10) 7. AUTHOR(s) Donald Dewey, Stephen Behnen, Richard Bousley, James Mason		(15) 8. CONTRACT OR GRANT NUMBER(s) F33615-77-C-1233
9. PERFORMING ORGANIZATION NAME AND ADDRESS Boeing Military Airplane Development, Boeing Aerospace Company, Seattle, Washington 98124 622 04F		(16) 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2003-01-14 (17) 01
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory (AAS), Wright-Patterson Air Force Base, Dayton, OH 45433		(11) 12. REPORT DATE May 79 13. NUMBER OF PAGES 164
(12) 787 p. 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

410 258

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)

18. SUPPLEMENTARY NOTES

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

SOFTWARE	DIGITAL AVIONICS INFORMATION SYSTEM
EXECUTIVE	NAVIGATION
AVIONICS	JOVIAL J-73

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

This effort has provided an independent contractor assessment of the DAIS executive software, industry exposure to the J-73 level one compiler and DAIS software development tools and a definition of the interface between DAIS and integrated navigation systems. A combination of INS, GPS and air data was used to demonstrate the usefulness of the DAIS/integrated navigation system for tactical air drop and terminal area operations w/transport aircraft. A →

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 258

Boeing Aerospace Lab Simulation jointly managed by AFAL and Boeing, was conducted for the purposes of evaluating the DAIS/integrated navigation system software performance capabilities in a simulated tactical air drop and terminal area navigation flight profile. A 'Hot Bench' simulation was conducted using contractor developed sensor software modules and navigation filter, and the DAIS executive software and processing hardware. Outputs of the program include: Verification of the overall DAIS concept, measurements of executive software overhead and suggestions for improving the DAIS executive software.

FOREWORD

This final report summarizes the results of contract F33615-77-C-1233, "Evaluation of DAIS Technology Applied to the Integrated Navigation System of a Tactical Transport". This document was prepared for the Air Force Avionics Laboratory by the Boeing Military Airplane Development (BMAD) division of The Boeing Company in Seattle, Washington. Contract managers for this program were Mr. Ken Normand, AFAL/AAA, followed by Mr. James Bain, AFAL/AAS.

The measurements and evaluations herein are based on 1977 versions of DAIS documentation and software, many of which have been subsequently revised. Consequently, some of the measurements and comments herein are not applicable to the current DAIS system. Detailed information on the current status of DAIS documentation/software can be obtained from the DAIS program office.

Table of Contents

Table of Contents Continued

NOTE: Due to the size limitations, Appendices C and D are not included in this document. These appendices may be obtained by writing the DAIS Library, AFAL/AAS, Wright-Patterson AFB, Ohio 45433, and requesting DAIS document #79-05.

List of Figures

List of Figures Continued

List of Tables

List of References

Document	Title
SA100101	System Segment Specification for DAIS, Appendix A, DAIS System Control Procedures - Functional Flow Diagrams, 7 Nov. 1975.
SA200201	System Control Procedures for the Digital Avionics Information System, 7 July 1977.
SA201302 PT I	DAIS Mission Software Exectuive Specification, 10 June 1976.
SA201302 PT II	DAIS Mission Software Product Specification, Vol. 1: Local Executive.
SA201302 PT II	DAIS Mission Software Product Specification, Vol. 2: Bus Control.
SA202200 PT II	PALEFAC Detailed Design Specification - Final, 15 Feb. 1977.
SA202201	PALEFAC Pre-Processor Detailed Design Specification - Final, 1 Feb. 1977.
SA204200 PT I	JOVIAL (J73/I) Language Specification, 19 Dec. 1974.
SA401301	DAIS Processor Instruction Set, 1 Oct. 1976.
SA802309C	PALEFAC Pre-Processor/PALEFAC to Mission Software Interface Control Document, 31 May 1977.
MA201200	DAIS System Control Procedures, 7 Mar. 1978.
MA202200	PALEFAC User's Guide, 1 Feb. 1977.
MA202200B	User's Manual for Partitioning, Analyzing, Linking, Editing Facility (PALEFAC), 1 May 1978.
MA204200-1	JOVIAL J73/I Computer Programming Manual, Oct. 1975.
MA206200-1	DAIS Processor/Cross Assembler User's Manual, 22 Dec. 1975.
MA206201	DAIS Assembler User's Reference Manual, 17 Feb. 1976.
MA207301	Performance Monitor and Control Executive User's Guide, 30 June 1977.

List of References Continued

Document	Title
MA212200	User's Manual for Software Test Stand Linker, 15 Feb. 1977.
MA401200-1	DAIS Processor Loading From DEC-10 (ASYTRN Program) User's Manual, 1 Nov. 1976.
PA200101	Software Development Standards, 15 Jan. 1976.
TA201302-2	DAIS Executive Software Efficiency Test, May 1979.
78-03	Test Plan for DAIS Executive Evaluation Program, Jan 1978.
79-01	Executive Evaluation Software Manual, May 1979.
79-02	Executive Evaluation User's Manual, May 1979.
79-05	Data Samples from DAIS Executive Evaluation, 1 July 1979.

All referenced documents are available from the DAIS Library.

List of Acronyms

ACTREG	Activity Register
ADA	New DOD language replacing J73
ADC	Air Data Computer
AFAL	Air Force Avionics Laboratory
ALAP	Avionics Lab Assembler Program
BCIU	Bus Control Interface Unit
BMAD	Boeing Military Airplane Development
CARP	Computed Air Release Point
CDC	Control Data Corporation
CDRL	Contract Data Requirements List
CIO	Console I/O to peripherals
CIU	Console Intelligence Unit
CPU	Central Processing Unit
DAIS	Digital Avionics Information System
DARTS	Digital Avionics Research Test System
DDS	Detailed Design Specification
DEC	Digital Equipment Corporation
DECSS	DARTS Environmental Control System Simulation
DINS	DARTS Integrated Navigation System
DOD	Department of Defense
EES	Executive Evaluation Software
EHAR	Error Handling And Recovery
ESO	Event Service Overhead
EX	Execute
GFE	Government Furnished Equipment
GPS	Global Positioning System
HBC	Hot Bench Computer
HOL	High Order Language
ICD	Interface Control Document
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
I/O	Input/Output
IPSR	Inter-Processor Service Request
ISO	Interrupt Service Overhead
JC	Jump on Condition
JS	Jump to Subroutine
LEO	Local Executive Overhead
LINKS	Software Test Stand Linker
MC	Minor Cycle
MCADU	Modest Control And Display Unit
MEO	Master Executive Overhead
MUX	Multiplex
OFP	Operational Flight Program
PAF	PALEFAC Auxiliary File
PALEFAC	Partitioning, Analyzing, Linking, Editing Facility

List of Acronyms Continued

PGI	PALEFAC Global Input
PINS	Pseudo-Integrated Navigation System
PMC	Performance Monitor and Control
PMI	PALEFAC Module Input
PP	Pre-Processor
PPT	PALEFAC PP Text Output
RT	Remote Terminal
SCADU	Super Control And Display Unit
SDS	Software Development Standards
SIL	Synchronous Instruction List
SRT	System Response Time
STOL	Short Take Off and Landing
STS	Software Test Stand
TCT	Test Control Table
TDT	Transmission Delay Time
URT	Universal Remote Terminal
WPAFB	Wright-Patterson Air Force Base

1.0 Introduction

1.1 Scope

This report summarizes the evaluation of the DAIS Executive Computer Program by the Avionics Technology group of the Boeing Military Airplane Development (BMAD) division of the Boeing Aerospace Company. This evaluation was conducted under Air Force contract number F33615-77-C-1233 in conjunction with ongoing avionics technology research programs at Boeing. The program was conducted between September 1977 and May 1979. The DAIS Executive Computer Program and support software which were evaluated were the current configurations made available by the Air Force Avionics Laboratory (AFAL) during December 1977.

This program was limited to an independent evaluation of the DAIS executive in terms of its application to a tactical transport mission. The overriding issues to be resolved in this program were (1) Can the DAIS Executive Computer Program support a tactical transport mission; and, (2) What is the cost of using the program in terms of computing resources and manpower? In answering these questions, a comparative analysis with other executives was not undertaken as part of the study. Rather, the DAIS Executive Computer Program performance was examined on its own merits, both parametrically (across the spectrum of avionics programs in general) and subjectively (in the context of a tactical transport integrated navigation program).

The performance measurement of the DAIS Executive Computer Program was accomplished in two phases. Phase I, the "Parametric Study," and Phase II, the "Verification Phase," are briefly described here and more fully described in paragraph 3 of this document.

During the Phase I parametric study the DAIS Executive Computer Program was instrumented with a number of software hooks which served as probes into the executive for performance data measurement. These software hooks are described in the "Test Plan For the DAIS Executive Evaluation and DAIS Executive Evaluation Software Design," sequence #1, CDRL attachment #2. Once the Executive Computer Program was instrumented with this test software, it was exercised with a controllable applications model, Pseudo-Integrated Navigation System (PINS), which was designed to simulate different avionic application programs, including a navigation program for a tactical aircraft. This programmable avionic software model was then operated in various modes, which were selected by varying in realtime a set of control parameters. Performance data was recorded for offline analysis. The offline analysis program isolated the effects of various executive control parameters and the overhead time for each executive parameter was computed. Then, using the computed parametric data, a composite predictive algorithm for executive performance was developed.

To verify the accuracy of predictive algorithm, the DAIS Executive Computer Program was instrumented in Phase II with the same executive evaluation software as described above. The PINS program which was used for the parametric study was replaced with the DARTS Integrated Navigation System (DINS), which had been developed by Boeing for tactical aircraft applications. The DAIS Executive Computer Program was then operated with the DINS program and performance data was recorded. The final step was to correlate the measured performance data with the predicted performance made during Phase I using the predictive algorithms.

During the development of both the PINS and DINS application programs, the DAIS Software Development Standards were followed. These standards were subjectively evaluated with regard to both their completeness and correctness when applied to a general applications program. In addition, the DAIS support software and its associated documentation were also appraised insofar as they affected the orderly and efficient development of a new avionics program.

1.2 Background

The Air Force is facing three basic problems in avionic program development: cost, reliability, and lack of standardization. High costs are being experienced in each phase of the development, procurement and support of new weapon systems. These high costs are escalated by the excessive development time required for an avionic system, a situation leading to major budgetary and scheduling restraints on new systems. In addition, poor field reliability has been a problem and the existing nonstandard avionics have proven difficult to repair. Lack of standardization has provided an environment where each new weapon system requires all new hardware, software and support equipment. These problems have been recognized by the Air Force, which is implementing an avionic strategy designed to reverse the trends. The basic USAF strategy is to establish goals for reducing avionic proliferation and cost, while increasing avionic availability.

One of the major programs in the Air Force is the Digital Avionic Information System (DAIS). Standards which are evolving through the development and use of the DAIS concept include the communication standard (MIL-STD-1553), the standard language JOVIAL J73 (MIL-STD-1589), the machine instruction set standard (MIL-STD-1750), and the executive interface standard. These three standards, when used together, can provide the environment for software transferability. The emerging executive interface standard is based on the executive applications interface developed by the DAIS program. This executive applications interface is an essential link in the chain of standards that will make it possible to transfer software programs from one avionic program to another.

Transferability and reusability of software is one important means of lowering both acquistion and life cycle costs. Software tasks written in a HOL (High Order Language) must communicate with the external environment and with other HOL tasks. This communciation is performed via realtime executive requests. The use of realtime statements in a language is not new. Realtime statements have been included in previous DoD languages (HAL/S, SPL/I) and will be included in the new DoD-1 language ADA. If the set of realtime executive requests is also standardized with a HOL, the interface between applications software and the executive will provide the ability to transfer software from one avionic system to another.

1.3 Program Objective

The DAIS Executive and Application Software Architecture Standards have been designed with the goal of providing the environment and rules to create modular and transportable avionic software. Before the DAIS Executive and Architecture Standards are applied to a major program, more information must be gathered about the utility of these proposed Air Force Standards. This study was conducted to collect this needed information. The specific questions which this study has addressed are:

(1) Is the DAIS executive concept applicable to general avionic software programs?

(2) If so, what is the cost (in terms of computer resources) of using the DAIS executive?

(3) Are the DAIS executive support software programs and standards adequately defined, and are there problems or inconsistencies within them?

(4) What changes and modifications are required before the DAIS executive and related support software and standards are used in full scale avionic development programs.

It was the objective of this program to answer the four questions above in the context of an avionic Operational Flight Program (OFP) software development program. Paragraph 2.0 of this report provides a summary of the program, including the approach taken to answer the questions as well as a summary of the results of the study. Paragraph 3.0 describes the methodology used to measure executive performance. Paragraph 4.0 presents the performance data collected and analyzed in summary form. Paragraph 5.0 discusses the DAIS support software and development standards used during the performance of this program, and recommendations for changes to each. Finally, paragraph 6.0 offers recommendations for changes to the DAIS executive.

2.0 Program Summary

The methodology used to meet the program objectives stated in paragraph 1.3 above, is described in paragraph 2.1. Additionally, the findings and conclusions of this study are briefly summarized in paragraph 2.2. The recommendations made for improving the DAIS executive, support software, and standards are summarized in paragraph 2.3.

2.1 Program Methodology

During this study, each component of the DAIS program that is used in the development of operational flight program (OFP) software was examined. These components are:

- o DAIS Executive Computer Program
- o DAIS Software Development Standards
- o DAIS Support Software

To meet the objectives of this program, it was necessary to apply these components to real avionic software programs, and not to simply study documentation and formulate opinions. For example, question 1 (paragraph 1.3), "is the DAIS Executive Computer Program applicable to general avionics programs," can only be answered by using the executive software in actual avionics applications. If applicable, then the costs of the computer resources required by the DAIS Executive Computer Program need to be determined by measuring these resources while the DAIS Executive Computer Program is operating in support of various OFP software programs. Similarly, realistic conclusions about the DAIS Software Development Standards and the DAIS Support Software can only be made by actually using them. To this end, operational software was designed to the interface guidelines defined in the DAIS Software Development Standards, and the DAIS Support Software was used to build load modules which operated under control of the DAIS Executive Computer Program. A summary of this program plan is illustrated in figure 2.1-1. The paragraph references inside the blocks refer to descriptions found in the program summary paragraphs that follow. The paragraph references outside the blocks refer to the detailed descriptions of the work performed.

2.1.1 Define Performance Parameters

Six executive performance parameters that are useful to a system designer were defined for measurement and evaluation. These parameters are defined in detail in the "Test Plan for the DAIS Executive Evaluation Program". In brief, the six parameters are:

1) Transmission Delay Time (TDT)

   This parameter is defined as the amount of time between a request for an asynchronous transmission and the actual time of the transmission.

2) Interrupt Service Overhead (ISO)

   This parameter is defined as the amount of time spent by the executive interrupt service routines servicing an interrupt.

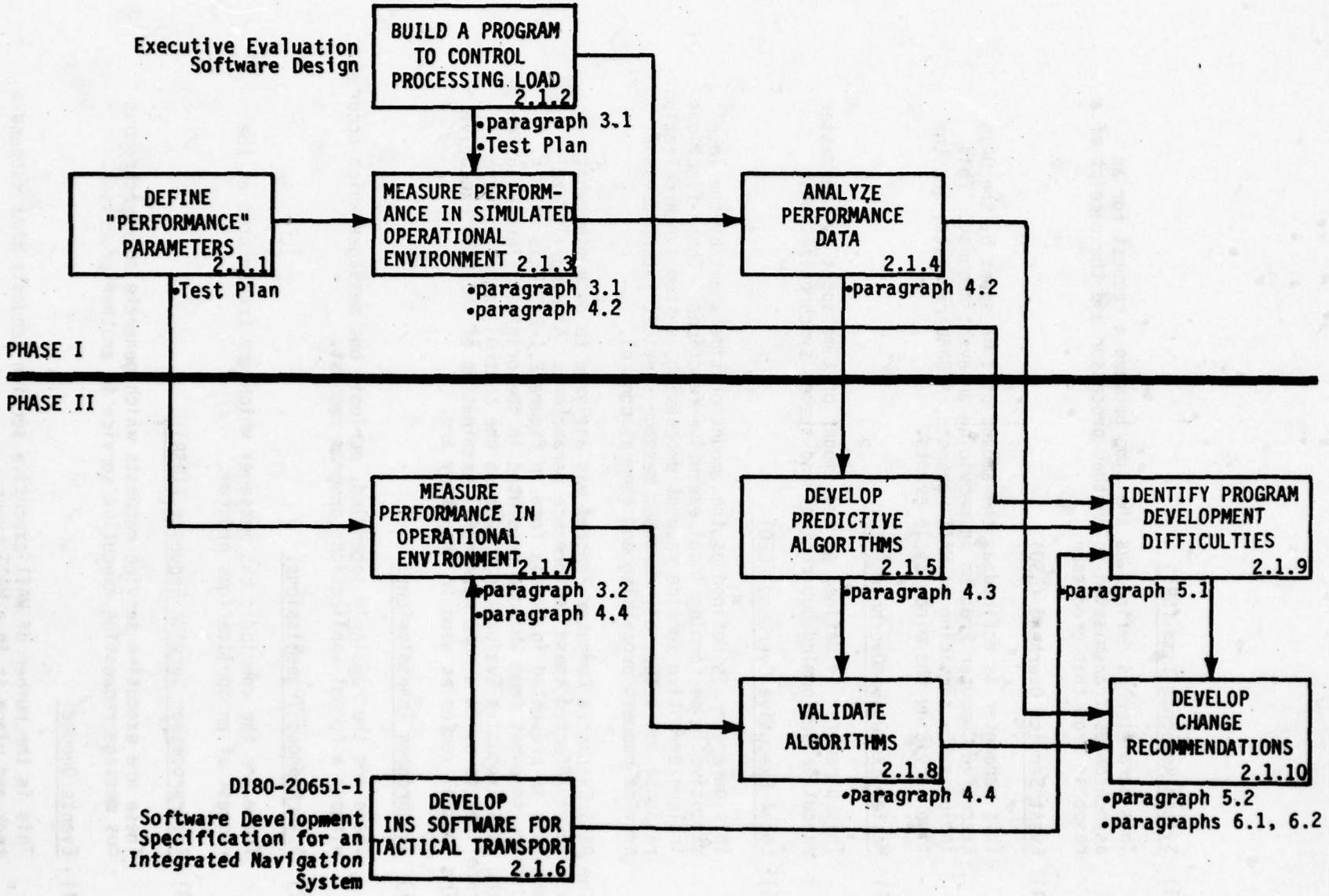


Figure 2.1-1 PROGRAM PLAN

3) System Response Time (SRT)

This parameter is defined as the time between a request for an asynchronous transmission to another processor and the receipt of a response from that processor.

4) Event Service Overhead (ESO)

This parameter is defined as the amount of time spent by the DAIS Executive Computer Program in servicing an event request. This includes the servicing of WAIT requests, SIGNAL requests, and the time servicing the minor cycle events.

5) Master Executive Overhead (MEO)

This parameter is defined as the amount of time spent by the master executive performing bus control and system synchronization.

6) Local Executive Overhead (LEO)

This parameter is defined as the amount of time spent by the local executive in performing local executive functions. These functions include executive service request processing, system synchronization response and setup, asynchronous message service, interprocessor service request processing and general control.

The DAIS Executive Computer Program was analyzed to determine what variables affected these performance parameters. A summary of this analysis is presented in matrix form in figure 2.1-2. This matrix differs somewhat from the one presented in the original "Test Plan for the DAIS Executive Evaluation," because the control variables were redefined to allow a more meaningful examination of the DAIS executive. The control variables used in the study are:

1) Synchronous Transmissions:

These are the regularly scheduled, periodic bus messages which occur without a formal applications program request.

2) Asynchronous Transmissions:

These are the aperiodic bus messages which are transmitted at the request of an applications program.

3) Interprocessor Service Requests (IPSR):

These are executive service requests which generate an asynchronous bus message requesting executive service in another processor.

4) Events Queued:

This is the number of WAIT executive service requests that suspend a task and place it in a WAIT queue.

CONTROL VARIABLES \ PERFORMANCE VARIABLES	TRANS-MISSION DELAY TIME	INTERRUPT SERVICE OVERHEAD	SYSTEM RESPONSE TIME	EVENT SERVICE OVERHEAD	MASTER EXECUTIVE OVERHEAD	LOCAL EXECUTIVE OVERHEAD
SYNCHRONOUS TRANSMISSIONS (SIL)	1		7			
ASYNCHRONOUS TRANSMISSIONS (NON-SIL)	2	4	8	11	14	17
INTERPROCESSOR SERVICE REQUESTS	3	5	9		15	18
EVENTS QUEUED (WAIT)		6		12		19
EXECUTIVE SERVICE REQUESTS (NON IPSR)						20
EVENTS SIGNALLED (MC, SIGNAL)			10	13	16	21

Legend: (a) shaded areas indicate relationships that were investigated
(b) numbers in shaded areas refer to "Notes" on page 8. These notes explain the relationships between the control variables and the performance variables

Figure 2.1-2 TEST PARAMETER MATRIX

5) Executive Service Requests (Non-IPSR):
These are requests for executive service which are satisfied in the local processor - no bus transmission is required.

6) Events Signalled

These are the requests which result in an event being set. These include both signalled events and minor cycle events.

Each of these control variables can affect some or all of the performance parameters defined for this study. Figure 2.1-2 shows this relationship. The affect a control variable has on a given performance parameter is outlined briefly in the list below, where the number of the item in the list refers to the numbers in the shaded areas of figure 2.1-2.

(1) May have a minor effect. The master executive is unable to recognize a need for an asynchronous transmission until a status word is received from the remote processor. If the SIL is being executed, a status word will not be requested until a message sequence is completed.

(2) Has a major effect if an asynchronous transmission is in the asynchronous transmission queue ahead of the transmission being requested. If so, the request will not be recognized until processing has been completed for the first transmission.

(3) Effect is the same as (2) above since an interprocessor service request is an asynchronous transmission.

(4) Every asynchronous transmission will cause one interrupt in the local executive of the processor where the request originates. Requests originating in a remote processor will also cause two interrupts in the master processor.

(5) Effect is the same as (4) above since an interprocessor service request is an asynchronous transmission.

(6) A WAIT specifying an absolute time interval will cause a timer interrupt when the wait period has expired.

(7) The SIL affects SRT in the same as manner described for TDT (1). For SRT, however, the effect can be magnified since access to the bus may be delayed both when obtaining the request from the remote processor and then when sending the response to the remote processor.

(8) Asynchronous transmissions affect SRT in the same fashion as described for TDT in (2). The effect may be greater for SRT, however, since previously queued asynchronous transmissions may have to be completed both before the request surfaces in the remote processor and before the response is sent from the master processor.

(9) The effect is the same as (8) above since an interprocessor service request is an asynchronous transmission.

(10) SRT measurements may extend over more than one minor cycle. In such cases, the number of minor cycle events serviced in the new minor cycle can affect SRT.

(11) Asynchronous transmissions may have an event associated with them.

(12) Every event which is set must be checked against the entries in the event wait queue.

(13) Every event requires servicing when it is signalled. This includes minor cycle event service.

(14) Every asynchronous transmission must be serviced by the master executive.

(15) Every IPSR must be serviced by the master executive.

(16) The master executive must send the minor cycle event (mode code 18) to each processor in the system.

(17) Every asynchronous transmission request requires service by the local executive, both when it is queued for transmission and when it is received.

(18) Every IPSR requires servicing in both the processor that generates it and in the processor that receives it.

(19) Executive service time is required to queue an event.

(20) Executive processing is required to service realtime statements executed in an applications program.

(21) Minor cycle setup (not including ESO) is handled by each local executive.

2.1.2 Build a Program to Control Processing Load

Once the control variables were isolated, a program was developed which would allow realtime control of these variables. Realtime control allowed measurement on DAIS executive performance while it was supporting a known system load. This program, the Pseudo-Integrated Navigation System (PINS) is described in "Executive Evaluation Software Design," attachment #2, CDRL #1.

2.1.3 Measure Performance in a Simulated Operational Environment

The DAIS executive was instrumented with software measurement subroutines which allowed elapsed times to be recorded for selected executive routines. The DAIS executive was then exercised with the PINS program and performance measurements were taken.

2.1.4 Analyze Performance Data

The performance data from paragraph 2.1.3 was analyzed to determine the effect of each of the control variables shown in figure 2.1-2 on executive performance parameters. This analysis completed the Parametric Study.

2.1.5 Develop Predictive Algorithms

Working from the analytical results of the Parametric Study, a set of predictive algorithms were developed that would provide performance estimates for the executive under any known processing load.

2.1.6 Develop INS Software for Tactical Transport

An applications software program was developed by Boeing using the DAIS Software Development Standards. This software, the DARTS Inertial Navigation System, integrates information from a Global Positioning System receiver, a strapdown Inertial Navigation System, and an Air Data Computer and passes it through a Kalman filter to provide corrected position and velocity data. The predictive algorithms were applied to DINS to predict DAIS executive performance for that applications program.

2.1.7 Measure Performance in an Operational Environment

The DAIS executive was instrumented with the software measurement routines used in Phase I (paragraph 2.1.3). The DAIS executive was exercised with the DINS program and performance measurements were taken.

2.1.8 Validate Algorithms

The performance data from paragraph 2.1.7 was analyzed. Good agreement between predicted performance and measured performance served to validate the predictive algorithms. The verification step of Phase II was completed.

2.1.9 Identify Program Development Difficulties

Two avionics programs, PINS and DINS, were designed according to the DAIS Software Development Standards and built using the DAIS Support Software. Any difficulties encountered during this process were documented with descriptions of both the problem that was encountered and the impact of the problem on the program development.

2.1.10 Develop Change Recommendations

Experience with the program development process uncovered several areas in the DAIS standards and documentation which could be changed to benefit a system designer. In addition, the results of the data analysis indicated that certain modifications to the design of the DAIS system could provide an overall improvement in system performance. These findings are presented as a series of change recommendations in paragraphs 5.0 and 6.0.

2.2 Program Findings

2.2.1 DAIS Executive Concept

The question of whether the DAIS Executive Computer Program was applicable to general avionics applications was answered early in the program. Based upon the experience gained from designing, building, and operating a general avionics application program (PINS), and an integrated navigation system for a tactical transport application (DINS), the DAIS executive, support software, and design standards were proven to be highly effective for avionics applications. Not only did the integrated navigation software perform properly under the DAIS executive control, but the development of the applications software was greatly simplified by having an executive program which was already developed and under configuration control, and by the rigid executive/applications interface structure.

In summary, an existing executive program was taken "off-the-shelf" and two separate avionics programs were made operational under control of this executive without making a single change to the executive. This experience demonstrates the significant advantages of a standard, general purpose executive for avionic applications.

2.2.2 DAIS Executive Performance (Computer Resource Cost)

2.2.2.1 General Performance

The cost in computer resources to operate the DAIS Executive Computer Program was found to be very reasonable. In fact, considering the general purpose nature of the program itself and the tasks being performed by the executive, the executive appears to be fairly efficient. For example, the steady-state overhead associated with the executive computer program was consistantly measured at approximately 27% for the master executive computer program and approximately 10% for the local executive computer program.

The steady state overhead reflects only bus control processing and minor cycle (system) synchronization. All other factors affecting the overhead (in addition to the steady state overhead) were found to be strictly dependent upon the level of requests generated by the applications program.

The performance parameters identified in figure 2.1-2 were measured for various values of the control variables. It was found that a change in each control variable resulted in a proportional change in the performance parameter being measured. This was also true when various combinations of control variables were changed simultaneously; the same delta effect was observed when each of the control variables were individually varied in three separate tests. These tests and the measured results are detailed and summarized in paragraph 4.2 of this report.

The data collected during Phase I of this study were used to develop a set of predictive algorithms for estimating DAIS executive overhead for any known applications load. Because the DAIS executive performance was linear, the development of predictive algorithms was straightforward. These algorithms, discussed in paragraph 4.3, were used to predict the executive computer program resources required to support the DINS application program for the second phase of the program. In Phase II, executive performance was measured while supporting the DINS program. The results of these measurements, presented in paragraph 4.4, show good agreement with the predictive algorithms.

2.2.2.2 Executive Service Request Overhead

In addition to predicting the total overhead that will be experienced by a particular software system design, the predictive algorithms also can be used to estimate the maximum number of executive services that can be used during a single DAIS minor cycle. These estimates are provided for each individual executive service under the assumption that the applications load in the processor is negligible. This data provides upper bounds to the system designer when allocating executive services to the application programs.

The DAIS architecture introduces two factors that determine the executive processing required when an applications program requests an executive service. The first factor is the location of the applications program that generates the request. If the applications program is in the master processor, less computer time is available to process executive service requests since the bus control services provided by the master executive must also be supported. The second factor is whether the executive service requires a bus transmission. A local executive service which requires no bus transmission can be completed in much less time than one which does. Combinations of these two factors result in four different levels of executive support that can be provided for the various executive services requested in applications tasks. Each of these levels will be discussed in turn.

The first level of support is for service requests handled by the local executive in a remote processor when no bus transmissions are required. The maximum number of any one of these local requests that can be serviced in one minor cycle is:

51	-	READs
26	-	WRITEs
77	-	SCHEDULEs
20	-	CANCELs
23	-	SIGNALs
94	-	Task Activations

The second level of support is for service requests handled by the local executive in the master processor when no bus transmissions are required. The maximum number of any one of these local requests that can be serviced in one minor cycle is:

35	-	READs
18	-	WRITEs
53	-	SCHEDULEs
13	-	CANCELs
16	-	SIGNALs
65	-	Task Activations

The third level of support is for service requests originating in a remote processor which require master executive support for a bus transmission. The maximum number of any one of these global requests that can be processed in one minor cycle is:

3	-	WRITEs
3	-	SCHEDULEs
3	-	CANCELs
3	-	SIGNALs

The last level of support is for statements which originate in the master processor and which require master executive support for a bus transmission. The maximum number of any one of these global requests that can be processed in one minor cycle is:

2	-	WRITEs
2	-	SCHEDULEs
2	-	CANCELs
2	-	SIGNALs

As can be seen, the use of any global request is very costly. These requests should be used with caution by a system designer to avoid "hot spots" or overload situations in system operation.

2.2.3 DAIS Executive Support Software and Standards

The DAIS Software Development Standards were found to be good in most areas but deficient in others. The overall standard, which includes several industry accepted standards of long-proven value, appears to be reasonable. In particular, the "top-down" design methodology and structured programming facets of the standards are quite good.

Areas where the standards might be improved are discussed in section 5.9 of this report. In summary, these areas included: design coding and methodology; program management and configuration control; sometimes overly restrictive implementation standards; and, vague or inconsistent standards.

The set of support programs necessary to build an operational applications program seems to be complete. During the study, operational systems were generated without building any additional support programs. The support programs were judged to be efficient and comprehensive. Problems that were encountered with these programs are almost exclusively related to weak documentation and immature diagnostic reporting capabilities. These findings are described in section 5.0 of this report.

2.3 Change Recommendations

The change recommendations being made as a result of this study fall into two categories. The first category covers suggested modifications to the DAIS Executive Computer Program and deals almost exclusively with error handling and recovery since this area is only partially defined within the executive computer program. Modification recommendations primarily address the rehosting and reusability of the program in other system architectures utilizing different hardware. The recommendations are described in paragraph 6.0 of this report.

The second category, which covers change recommendations for the DAIS Software Development Standards and Support Software deals almost exclusively with documentation corrections and clarification. These recommendations are described in paragraph 5.0 of this report.

3.0 Data Collection Methodology

3.1 Phase I

Phase I of the DAIS Executive Evaluation Study was designed to quantitatively measure the performance of the DAIS executive under a controlled load. The hardware and software configurations used for this task, the test procedures followed, the operating environment experienced by the executive, and the offline analysis software developed for the study are all discussed in the paragraphs that follow.

3.1.1 Hardware Configuration

The hardware configuration is shown in figure 3.1-1. The hardware necessary to support Phase I is listed in figure 3.1-2 along with the software that is resident in each processor unit. In Phase I, the Harris/6 performs in a stand-alone mode; there is no BCIU to connect it to the MIL-STD-1553A bus.

3.1.2 Software Configuration

The software that is resident in each processor is shown in figure 3.1-2. Note that processors which show more than one software element do not necessarily host all elements at once. For example, the PMC software and the Instrumentation Software cannot coexist in the PDP 11/40. Brief descriptions of the functions performed by each software element are given in figure 3.1-3.

3.1.3 Test Procedure and Data Flow

A brief description of the test procedure used in Phase I follows. For a more detailed discussion of the Phase I test procedure refer to "Test Plan for the DAIS Executive Evaluation", or to the "User's Manual for the DAIS Executive Evaluation Study."

a) Select the test to be run
   1. Choose the executive to be investigated: master executive, local executive in the master processor, or local executive in the remote processor.
   2. Select the test to be instrumented: interrupt service overhead, transmission delay time, etc.
   3. Select a test control table.

b) Power up the system
c) Load the processors
d) Instrument the hooks with the Executive Evaluation Software
e) Ready the system for the test
f) Run the test and collect the data
g) Process the test data
h) Power down the system

The data flow associated with the Phase I tests is shown in figure 3.1-4.

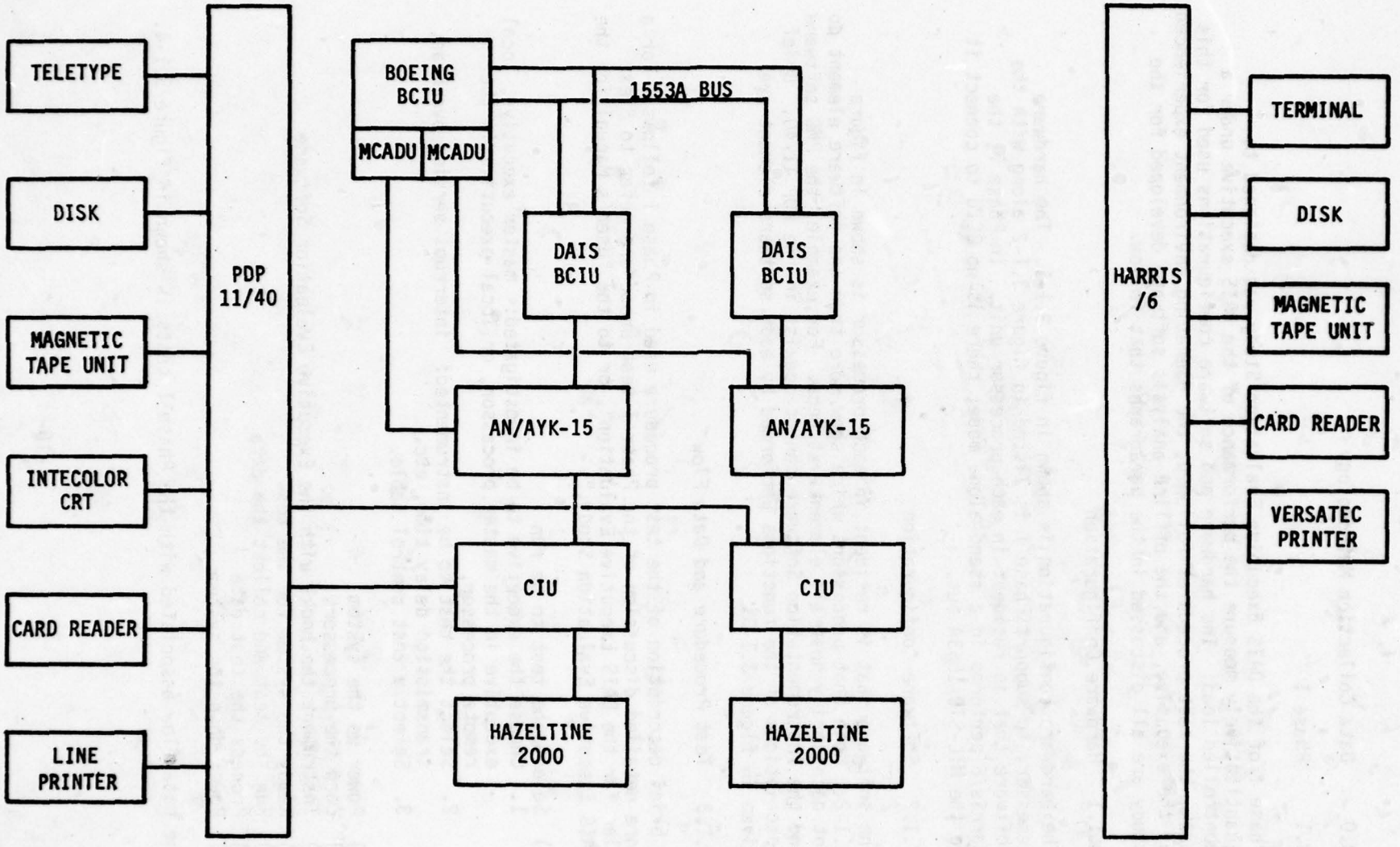


Figure 3.1-1 PHASE I HARDWARE CONFIGURATION

HOST HARDWARE	RESIDENT SOFTWARE
AN/AYK-15 Master Processor	DAIS Master Executive DAIS Local Executive Pseudo-Integrated Navigation System Executive Evaluation Software
AN/AYK-15 Remote Processor	DAIS Local Executive Pseudo-Integrated Navigation System Executive Evaluation Software
DAIS Bus Control Interface Unit (2 units)	Hardwired Logic BCIU Microcode
DAIS Console Intelligence Unit (2 units)	Firmware
Hazeltine 2000 CRT (2 units)	None
Boeing Bus Control Interface Unit Boeing Programmable Test Set	BCIU Macro Code (Programmed to be compatible with MIL-STD-1553/DAIS operation)
Modest Control and Display Unit (2 units)	Firmware
PDP 11/40 Computer Teletype Card Reader Line Printer RK05 Disk Drive (2 units) Magnetic Tape Unit Intecolor 8001 CRT	Performance Monitor and Control Instrumentation Software
Harris/6 Computer Silent 700 Terminal Card Reader Versatec Printer/Plotter 5260A Disk Drive Magnetic Tape Unit	Offline Analysis Program

Figure 3.1-2 Phase I Software Location

SOFTWARE	MAJOR FUNCTIONS
DAIS Master Executive	Control bus communications Process timer interrupts Handle startup and reconfiguration
DAIS Local Executive	Process realtime statements in applications tasks Control task states
Performance Monitor and Control (PMC)	Load the AN/AYK-15 processors
Pseudo-Integrated Navigation System (PINS)	Introduce a controlled load on the DAIS executives
Executive Evaluation Software (EES)	Generate MCADU records before and after certain executive processing sequences
BCIU Macro Code	Configure the Boeing BCIU to simulate a DAIS RT
Instrumentation Software	Transmit data blocks to and receive data blocks from the DAIS processors Record bus and MCADU data on tape
Offline Analysis Program	Retrieve bus and MCADU data from tape Generate statistical summaries

Figure 3.1-3 Phase I Software Functions

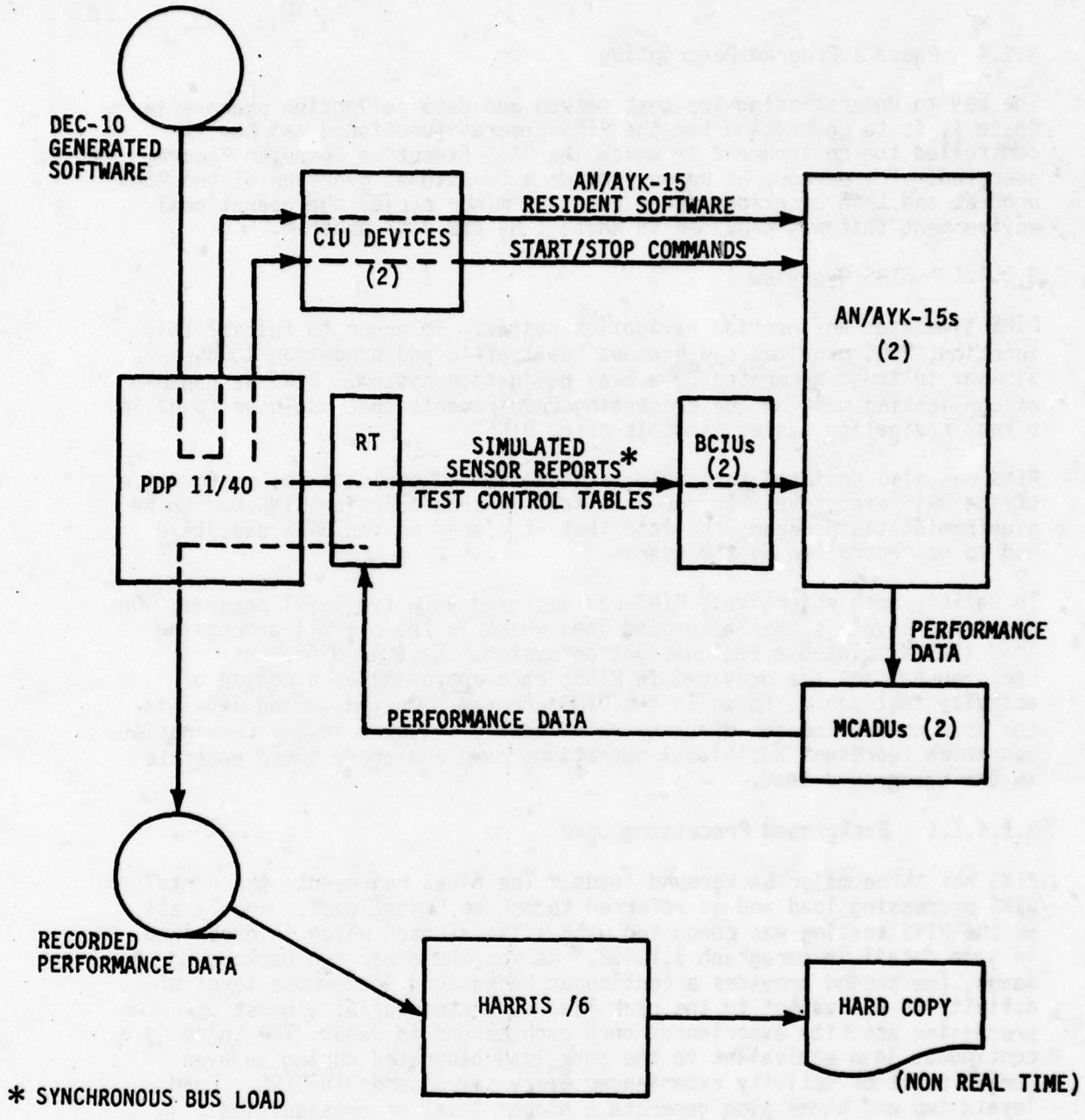


Figure 3.1-4 PHASE I DATA FLOW

3.1.4 Phase I Program Description

The key to understanding the test method and data collection process in Phase I, is to understand how the PINS program functioned and how it controlled the environment in which the DAIS Executive Computer Program operated. The paragraphs below provide a functional overview of the PINS program and then describe, minor cycle by minor cycle, the operational environment that was provided in Phase I by the PINS program.

3.1.4.1 PINS Overview

PINS simulates an inertial navigation system. In order to fulfill this function, PINS provides synchronous bus traffic and processor loads similar to those generated by a real navigation system. PINS is capable of duplicating most of the processing requirements that would be found in a real navigation system (in this case, DINS).

PINS was also designed to provide a mechanism to evaluate the performance of the DAIS executive. In order to fulfill this function PINS had to be a deterministic program; the load that it placed on the DAIS executive had to be controlled by the user.

To satisfy both objectives, PINS was designed as a two-level program. On the first level is the background load which is the overall processing load that simulates a real navigation system. Several different background loads are provided in PINS; each approximates a degree of activity that can be found in the DINS program. On the second level is the user controlled set of tasks which can be selected in any combination and which represent additional operations over and above those generated in the background load.

3.1.4.1.1 Background Processing Load

PINS has three major background loads.* The first represents the normal DINS processing load and is referred to as the "base load". Nearly all of the PINS testing was conducted with a "base Load" which is described in some detail in paragraph 3.1.4.2. As for the other two background loads, the second provides a continuous background load whose level of activity is equivalent to the peak load generated duri g a burst of processing activity experienced once each second in DINS. The third is a continuous load equivalent to the peak load generated during an even larger burst of activity experienced every six seconds in DINS. Load levels two and three also generate a higher level of prespecified asynchronous bus traffic.

The base load is an ordered sequence of tasks that is spread over eight minor cycles. This eight minor cycle sequence repeats itself during a test, duplicating the characteristic processing cycle found in DINS. As a result of the eight minor cycle period, any task operating in the base load will be executed a multiple of 16 times a second. Since each test phase lasts exactly one second, a minimum of 16 data points will be produced for any routine that is monitored.

*The background loads are described in detail in the Executive Evaluation Software Manual, DAIS document #79-01, pp186-191.

A special feature of the PINS program is its wait tasks.* Each processor contains a low priority task which is activated only when all other tasks have been completed. Each of these wait tasks contains a software clock which increments every 100 usec. By reading this clock at the beginning and end of a test phase it is possible to determine how much time was spent in executing the wait task. Since the wait task is executing only when the processor would otherwise be idle, the wait clock provides data on how much "unused" time was available during the test phase. In effect, the wait clock monitors how long the executive is "waiting" for something to do.

Each test run on the DAIS executive during Phase I began with a calibration phase. This phase consisted of just the base load; no extra tasks were performed. Execution of the calibration phase provided a benchmark which allowed the user to verify that the basic system performance remained constant from one test to the next.

3.1.4.1.2 User Controlled Load

The PINS program allows the user to select specific realtime statements to be executed in addition to the selected background load. These realtime statements include SCHEDULE, CANCEL, READ, WRITE, SIGNAL, and TRIGGER, and their execution is under the complete control of the user, both in numbers and in minor cycle of execution. The user can select a different set of these statements to execute during each phase of the test. Since the PINS background loads operate in an eight minor cycle repeating sequence, the user controlled load is also designed to repeat every eight minor cycles. The user has only to specify the first minor cycle in which an "event" will occur and the event will automatically occur every eight minor cycles thereafter. For any single "event" (for example, a SIGNAL) the user can cause the event to occur in as many minor cycles as desired. In addition, the user can control how many times the event will occur in a single minor cycle. For example, in one test phase the user may wish to observe the effect of adding one SIGNAL to minor cycle 3 (and 11, 19, 27, . . ., 123). In the next phase the user can look at the effects of adding two SIGNALs in both minor cycle 3 and minor cycle 6. The following phase can instead add TRIGGERs to the minor cycles since each test phase is completely independent of the others. Every realtime statement may be selected by minor cycle, with the restriction that a READ and WRITE or a SCHEDULE and CANCEL must occur in pairs. To demonstrate the flexibility of this system, the following example is provided. A user may design a test phase which adds a SCHEDULE and CANCEL to minor cycle 0, two READs and WRITEs to minor cycle 2, a SIGNAL to minor cycle 3, a TRIGGER to minor cycle 4, and two READs and WRITEs to minor cycle 6. This entire sequence of events will be repeated every eight minor cycles for the duration of the test.

3.1.4.2 Operational Environment

The PINS base load can be delineated by minor cycle to enable the user to determine the exact processing tasks being performed at any time during a test. Figure 3.1-5 provides this information.

*The PINS "wait" task should not be confused with executive "WAIT" request mentioned on page 6.

Several observations can be made about the PINS base load. In the master processor the PINS master configurator is activated once each minor cycle and in the remote processor the PINS remote configurator is activated once each minor cycle. This is in general agreement with the DINS execution sequence which has one or more tasks being activated nearly every minor cycle.

PINS synchronous bus activity occurs in only two minor cycles; zero and five. This agrees with the synchronous transmission activity found in DINS. All PINS synchronous bus transmissions repeat 16 times a second.

PINS models only the DINS synchronous bus transmissions which occur at intervals of less than one second since a PINS test phase lasts for only one second. None of the DINS one second transmissions are modeled in PINS.

PINS receives two synchronous inputs and transmits four synchronous outputs every eight minor cycles. Six privileged mode tasks support this synchronous activity. Two of the tasks execute in the remote processor during minor cycle one to access the synchronous data blocks received by the remote processor in minor cycle zero. Another pair of privileged mode tasks execute in the remote processor during minor cycle one to BROADCAST two of the synchronous data blocks that will be transmitted in minor cycle five. The final pair of privileged mode tasks execute in the master processor during minor cycle four to BROADCAST the other two synchronous data blocks that will be transmitted in minor cycle five.

The execution of individual events during the PINS processing sequence is controlled by a user defined Test Control Table. This table must be read each time the master and remote configurators are activated. Thus, as a part of the base load, PINS executes a READ once each minor cycle in each processor. Even though this READ is required only to support PINS, it still follows the general DINS activity level since DINS executes one or more READ statements in many minor cycles.

The final activity performed in the PINS base load sequence is an asynchronous WRITE and two READs. The asynchronous WRITE and one READ execute in minor cycle six; the other READ executes in minor cycle two. This asynchronous activity represents processing that could be expected upon receipt of an operator action from a keyboard.

In summary the PINS base load consists of the following tasks:

a. Master Processor
   (1) Activate master configurator every minor cycle
   (2) Read the Test Control Table (TCT) every minor cycle
   (3) Read a COMPOOL block in minor cycle two
   (4) Activate two privileged mode tasks in minor cycle four
   (5) Broadcast two COMPOOL blocks in minor cycle four
   (6) Allow the synchronous transmission of two COMPOOL blocks in minor cycle five
   (7) Perform a READ and an asynchronous WRITE in minor cycle six

Minor Cycle	Master Processor	Remote Processor
0	Activate Master Configurator Read Test Control Table	Activate Remote Configurator Read Test Control Table Receive two synchronous COMPOOL blocks via BCIU
1	Activate Master Configurator Read Test Control Table	Activate Remote Configurator Read Test Control Table Activate four privileged tasks Access two COMPOOL blocks Broadcast two COMPOOL blocks
2	Activate Master Configurator Read Test Control Table Read a COMPOOL block	Activate Remote Configurator Read Test Control Table Read a COMPOOL block
3	Activate Master Configurator Read Test Control Table	Activate Remote Configurator Read Test Control Table
4	Activate Master Configurator Read Test Control Table Activate two privileged tasks Broadcast two COMPOOL blocks	Activate Remote Configurator Read Test Control Table
5	Activate Master Configurator Read Test Control Table BCIU accesses two COMPOOL blocks	Activate Remote Configurator Read Test Control Table BCIU accesses two COMPOOL blocks
6	Activate Master Configurator Read Test Control Table Read a COMPOOL block Write COMPOOL block to Remote processor (Asynchronous)	Activate Remote Configurator Read Test Control Table Read a COMPOOL block Write COMPOOL block to Master processor (Asynchronous)
7	Activate Master Configurator Read Test Control Table	Activate Remote Configurator Read Test Control Table

Figure 3.1-5 PINS Base Load Processing by Minor Cycle

b. Remote Processor
   (1) Activate remote configurator every minor cycle
   (2) Read the Test Control Table (TCT) every minor cycle
   (3) Receive two synchronous COMPOOL blocks in minor cycle zero
   (4) Activate four privileged mode tasks in minor cycle one
   (5) Access two COMPOOL blocks in minor cycle one
   (6) Broadcast two COMPOOL blocks in minor cycle one
   (7) Read a COMPOOL block in minor cycle two
   (8) Allow the synchronous transmission of two COMPOOL blocks in minor cycle five
   (9) Perform a READ and asynchronous WRITE in minor cycle six.

Whenever the scheduled activity is completed for a minor cycle, the wait tasks resume execution and the wait clock is incremented every 100 usec until either the next minor cycle begins or executive action is requested.

3.1.5 Offline Analysis

The data collected during PINS test runs was analyzed offline on the Harris/6 computer. The offline analysis software is a FORTRAN program which reads the PINS test data recorded by the PDP 11/40, differentiates bus records from MCADU records, translates the records into human readable format, and performs data reduction on the MCADU records. The user can select a number of operational modes which include:

a. Dump the test data in octal format

b. Print the bus records recorded from bus number one

c. Print the bus records recorded from bus number two

d. Print all bus records

e. Print MCADU records

f. Pair MCADU start records with MCADU stop records and print the matched pairs

g. Pair the MCADU records, separate the pairs by type of event recorded, and perform data reduction on the separated sets.

The program mode used most often during this study was the last one. An example of the output obtained from mode g is shown in figure 3.1-6. Some of the information that can be found on this output is:

a. The processor, master or remote, from which the data was collected.

b. The "event" for which the statistics were compiled (e.g. the label in figure 3.1-6 identifies M$BCON activity following an interrupt 10). The event types are defined in DAIS document #79-05 which also contains Appendices C & D of this report.

c. The phases and minor cycles for which the statistics were compiled (the example in figure 3.1-6 indicates that statistics were collected for all test phases, but only data from minor cycle 3 of each test phase was examined).

d. The data for each test phase, presented on separate lines.

e. The number of events which were recorded during a given test phase.

f. The minimum and maximum elapsed times recorded for the "event" in each test phase. Each unit represents 100 microseconds.

g. The average time taken to complete the "event" in each test phase. Each unit represents 100 microseconds.

h. The percentage of time that each processor was being used to support both the executive and applications tasks.

The elapsed times recorded for each event are the sum of the event processing time plus the time spent in collecting the data (that is, the instrumentation hook time). It takes two instrumentation hooks to trap an event and record the elapsed time for analysis, one hook to record the time at the start of the event and a second to record the time when the event is completed. The processing time required for one of these hooks shows up in the recorded data. The time necessary to complete the second hook is not included in the recorded event time, but it does show up as an increase in CPU use. As a result of the instrumentation hook overhead, all of the event times shown in the reduced data will include the processing time for one hook. The final results, which will be used for computing executive overhead, are corrected for the instrumentation hook time and can be used with no further modification. Both the corrected and uncorrected data are listed in paragraph 4.0.

The data obtained in these tests were acquired by reading the DAIS processor clock known as timer B. This clock has a resolution of 100 usec. The other processor clock, timer A, has a 10 usec resolution, but its usefulness with respect to obtaining elapsed times is limited because the executive software can reset timer A (timer B is never reset). In all DAIS processors, timer A is reset by the executive software at the beginning of each minor cycle. Since some of the events measured could extend over more than one minor cycle, timer B had to be used to time them. In the master processor, timer A is also reset by the master executive whenever a TRIGGER is queued in the system. Since many of the tests in the master processor studied event processing while TRIGGERS were queued, timer B had to be used. To determine if timer B could be substituted for timer A in all cases, tests were run with both timer A and timer B. It was found that if the sample size was increased for the timer B measurements, results from using the two different timers were the same. Since thousands of samples were obtained for most of the events studied, timer B was used for all event timing.

3.2 Phase II

Phase I determined the individual times required by the DAIS executive to service the various requests generated by application software programs designed following the DAIS software development standards.

(a) MASTER PROCESSOR
STATISTICS FOR EVENT TYPE: 000140 000012 (96. 10.) (b)

(c)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
PHASE		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CYCLE	0	0	0	1	0	0	0	0																																	

(d) PHASE	ICOUNT	(e) N	--SUMMATION-- X	X SQRD	(f) ----X--- MIN	MAX	-VALID DATA- MIN	MAX	(g) MEAN	STD DEVIATION MEAN	1 EVENT	(h) ----CPU USE---- MASTER	REMOTE	
1	1176	16	61	235	3	4	2.5	5.1	3.81	.10	.40	47.0	24.3	1
2	2722	16	60	230	3	5	1.9	5.6	3.75	.14	.58	54.4	25.1	2
3	4106	16	59	223	3	5	1.7	5.7	3.69	.15	.60	53.3	25.8	3
4	5419	16	61	237	3	5	2.0	5.6	3.81	.14	.54	50.2	25.5	4
5	6732	16	60	228	3	4	2.3	5.2	3.75	.11	.45	50.3	25.2	5
6	7981	16	58	214	3	4	2.0	5.3	3.63	.13	.50	47.0	25.2	6
7	9506	16	59	223	3	5	1.7	5.7	3.69	.15	.60	54.4	25.1	7
8	10854	16	60	228	3	4	2.3	5.2	3.75	.11	.45	52.3	25.8	8
9	12168	16	62	242	3	4	2.8	5.0	3.88	.09	.34	50.3	25.5	9
10	13481	16	59	221	3	4	2.1	5.3	3.69	.12	.48	50.3	25.2	10
11	14734	16	61	235	3	4	2.5	5.1	3.81	.10	.40	47.0	25.2	11
12	16277	16	59	221	3	4	2.1	5.3	3.69	.12	.48	54.0	25.1	12
13	17631	16	58	214	3	4	2.0	5.3	3.63	.13	.50	51.8	25.8	13
14	18945	16	58	214	3	4	2.0	5.3	3.63	.13	.50	50.2	25.5	14
15	20258	16	59	221	3	4	2.1	5.3	3.69	.12	.48	50.0	25.1	15
16	21511	16	58	214	3	4	2.0	5.3	3.63	.13	.50	47.0	25.2	16
17	23054	16	62	242	3	4	2.8	5.0	3.88	.09	.34	54.1	27.4	17
18	24404	16	62	242	3	4	2.8	5.0	3.88	.09	.34	51.9	28.1	18
19	25718	16	58	216	3	5	1.6	5.7	3.63	.15	.62	50.2	27.8	19
20	27031	16	63	251	3	5	2.5	5.4	3.94	.11	.44	50.0	27.5	20
21	28284	16	61	237	3	5	2.0	5.6	3.81	.14	.54	47.0	27.6	21
22	0	0	0	0	0	0	0.0	0.0	0.00	0.00	0.00	-1.0	-1.0	22
23	0	0	0	0	0	0	0.0	0.0	0.00	0.00	0.00	-1.0	-1.0	23
24	0	0	0	0	0	0	0.0	0.0	0.00	0.00	0.00	-1.0	-1.0	24
25	0	0	0	0	0	0	0.0	0.0	0.00	0.00	0.00	-1.0	-1.0	25
26	0	0	0	0	0	0	0.0	0.0	0.00	0.00	0.00	-1.0	-1.0	26

Figure 3.1-6 Sample Output from Offline Analysis Program

NOTE: This is a sample of the data from Appendices C & D.

Phase II of the DAIS Executive Evaluation Study was designed to test the validity of the results obtained from Phase I by using the parametric results of Phase I to predict the executive overhead in an actual application program. The predicted overhead would then be compared to the measured overhead and if the measurements were the same, the Phase I results would be validated. The hardware and software configurations used for this task, the test procedures followed, the operating environment experienced by the executive, and the offline analysis software developed for the task are all discussed in the paragraphs that follow.

3.2.1 Hardware Configuration

The hardware configuration is depicted in figure 3.2-1. The hardware necessary to support Phase II is listed in figure 3.2-2 along with the software that is resident in each processor unit. In Phase II, the Harris/6 performs both in a stand-alone mode and in a configuration that links the Harris/6 into the system through the MIL-STD-1553A bus. When the navigation system is executing in the AN/AYK-15 processors, the Harris/6 is connected to the MIL-STD-1553A bus through a second Boeing BCIU. During offline analysis, the Harris/6 operates in a stand-alone mode to process the data collected on magnetic tape.

3.2.2 Software Configuration

The software that is resident in each processor is shown in figure 3.2-2. Note that the Harris/6 and the PDP 11/40 do not simultaneously support the two programs that run in them; only one software element is resident at a time. Brief descriptions of the functions performed by each software element are given in figure 3.2-3.

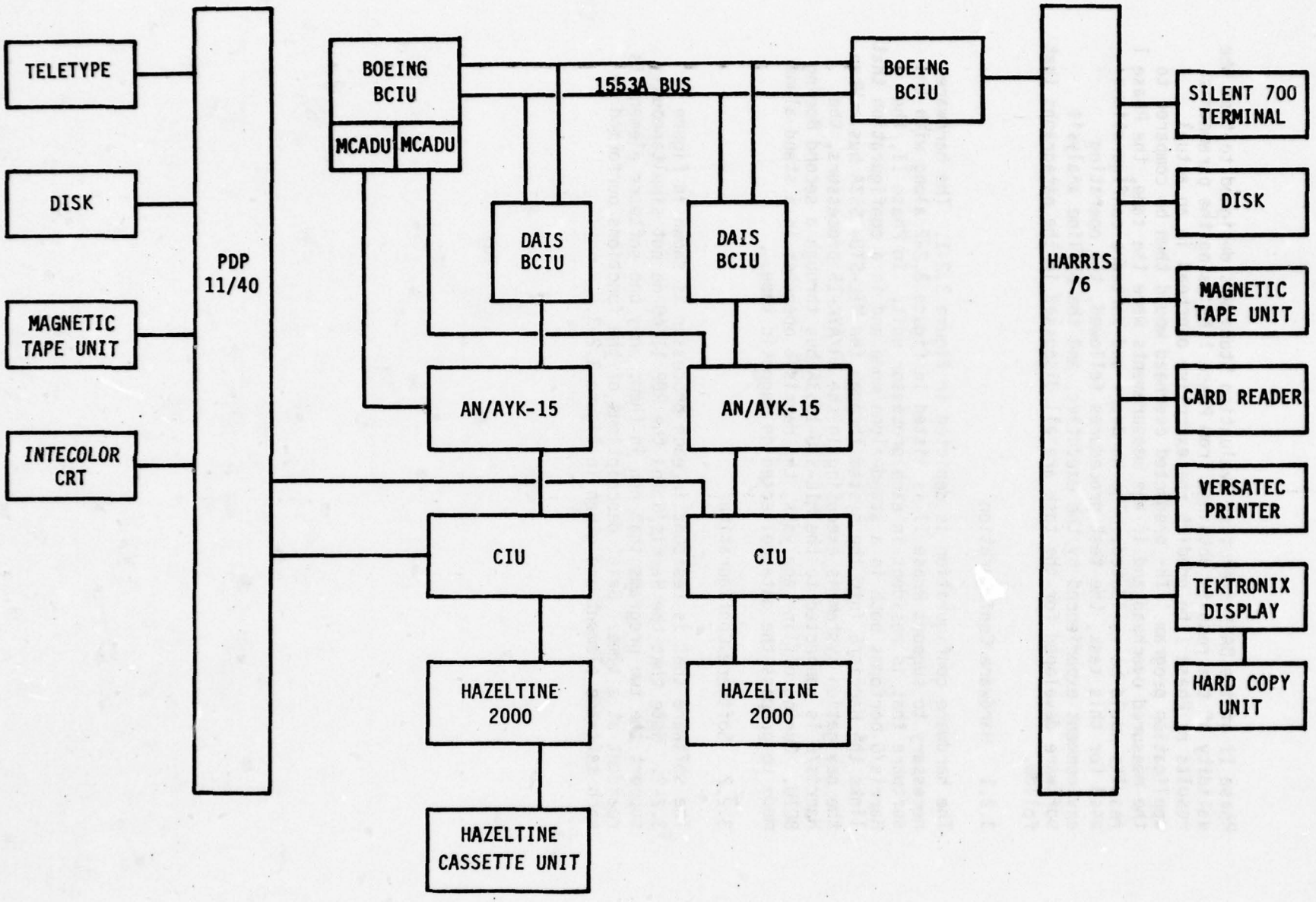


Figure 3.2-1 PHASE II HARDWARE CONFIGURATION

HOST HARDWARE	RESIDENT SOFTWARE
AN/AYK-15 Master Processor	DAIS Master Executive DAIS Local Executive DARTS Navigation System Executive Evaluation Software
AN/AYK-15 Remote Processor	DAIS Local Executive DARTS Navigation System Executive Evaluation Software
DAIS Bus Control Interface Unit (2 units)	Hardwired Logic BCIU Microcode
DAIS Console Intelligence Unit (2 units)	Firmware
Hazeltine 2000 CRT (2 units) Hazeltine Cassette Unit	None None
Boeing Bus Control Interface Unit Boeing Programmable Test Set	BCIU Macro Code (Programmed to be compatible with MIL-STD-1553/DAIS operation)
Modest Control and Display Unit (2 units)	Firmware
PDP 11/40 Computer Teletype RK05 Disk Drive Magnetic Tape Unit Intecolor 8001 CRT	Performance Monitor and Control Instrumentation Software
Harris/6 Computer Silent 700 Terminal Card Reader Versatec Printer/Plotter 5260A Disk Drive Magnetic Tape Unit 4014 Tektronix Display Tektronix Hard Copy Device	DARTS Environmental Control System Simulation Offline Analysis Program

Figure 3.2-2 Phase II Software Location

SOFTWARE	MAJOR FUNCTIONS
DAIS Master Executive	Control bus communications Process timer interrupts Handle startup and reconfiguration
DAIS Local Executive	Process realtime statements in applications tasks Control task states
Performance Monitor and Control (PMC)	Load the AN/AYK-15 processors
DARTS Integrated Navigation System (DINS)	Accept sensor inputs, use a Kalman filter to update position, velocity, etc., output corrections to the sensors
Executive Evaluation Software (EES)	Generate MCADU records before and after certain executive processing sequences
BCIU Macro Code	Configure the Boeing BCIU to simulate a DAIS RT
Instrumentation Software	Record bus and MCADU data on tape
Offline Analysis Program	Retrieve bus and MCADU data from tape Generate statistical summaries
DARTS Environmental Control System Simulation (DECSS)	Advance an aircraft along a preset trajectory, generate corresponding sensor inputs to DINS, monitor DINS outputs and calculate errors

Figure 3.2-3 Phase II Software Functions

3.2.3 Test Procedure and Data Flow

A brief description of the test procedure used in Phase II follows. For a more detailed discussion refer to "Test Plan for the DAIS Executive Evaluation."

a) Select the test to be run
   1. Choose the executive to be investigated: master executive, local executive in the master processor, or local executive in the remote processor.

   2. Select the test to be instrumented: interrupt service overhead, master executive overhead, etc.

   3. Determine which DINS mode (i.e., execution sequence and processor load) will be investigated.

b) Power up the system
c) Load the processors
d) Install the instrumentation hooks
e) Ready the system for the test
f) Run the test and collect the data
g) Load the Harris/6
h) Process the test data
i. Power down the system

The data flow associated with the Phase II tests is shown in figure 3.2-4.

3.2.4 Phase II Program Description

The DARTS Inertial Navigation System (DINS) is the navigation portion of an avionics software system. The paragraphs below describe the task execution sequence and processing load that was generated by DINS and measured in Phase II.

3.2.4.1 DINS Overview

The DINS software employs a mixing of navigation information from a Global Positioning System (GPS) receiver and a strapdown Inertial Navigation System (INS). Additional information is provided by an Air Data Computer (ADC).

In the Phase II environment, the DECSS executing on the Harris/6 supplies all of the above sensor information to DINS through a Boeing BCIU. This BCIU transmits data to the DINS and accepts sensor correction and display data from the DINS for transmission back to the Harris/6 computer. The BCIU operates as a remote terminal under control of the DAIS master executive. This data flow comprises the DINS synchronous bus traffic. It operates on a one-second cycle and remains at a constant level throughout the DINS execution.

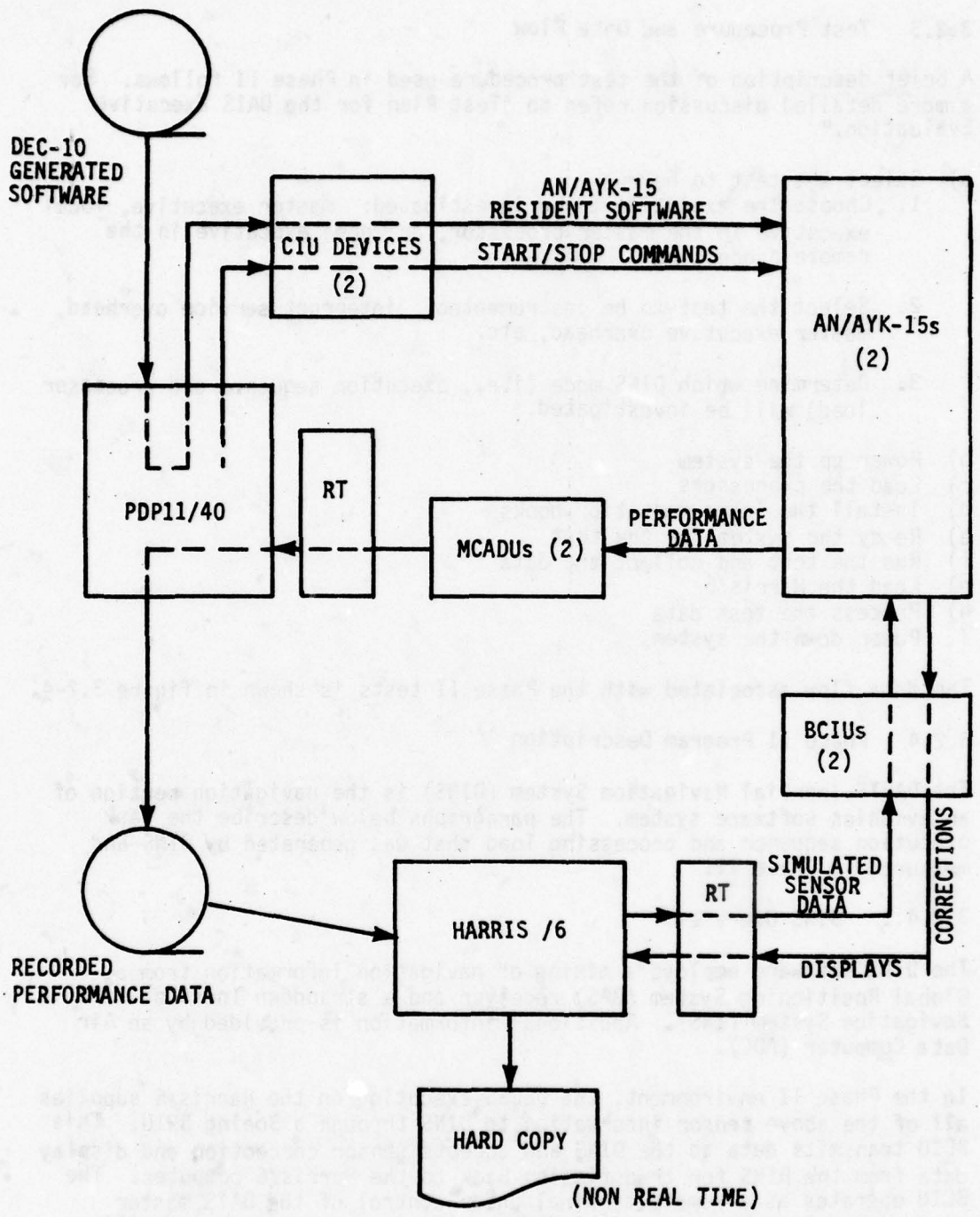


Figure 3.2-4 PHASE II DATA FLOW

3.2.4.2 DINS Operational Environment

In the DARTS Inertial Navigation System, the GPS is used to calibrate the error sources inherent to the INS and ADC. This is accomplished by using a Kalman filter. The calibration of INS error sources allows the INS to navigate in a free inertial mode when the GPS no longer provides information.

The GPS provides information to the DINS once in each Kalman filter cycle (6.0 seconds), so the full DINS execution sequence can best be described over a six-second cycle. In figure 3.2-5 the first second of the DINS cycle is shown, including the tasks executed and the number of executive service requests generated in each minor cycle. All tasks shown are resident in the remote processor (during the normal GPS-aided mode, no applications tasks are executing in the master processor). The last five seconds of the DINS cycle is outlined in figure 3.2-6. In both figures, the following task naming convention applies:

QPxx	Equipment handlers; move data between I/O COMPOOL blocks and navigation COMPOOL blocks
SP3x	GPS modules
SP4x	INS modules
SP5x	Kalman filter modules
DPxx	Data gathering modules for formatting display output to Harris/6
SP14	Software clock module for signalling six-second cycles

The READ and WRITE synchronous executive service requests shown in the figures were for an average block size of 13 words; each of these blocks had only one copy (in the remote processor). The asynchronous transmissions serviced compool blocks with an average length of 17 words; two copies existed for each compool block transmitted asynchronously, one in the master processor and one in the remote processor.

3.2.5 Offline Analysis

In order to measure the performance of the DINS program, it was necessary to instrument it with the same hooks that were used to collect the PINS data. In addition, the wait tasks were added to each processor to measure the otherwise unused execution time. The only difference between the data collection procedure for the two programs was that DINS did not use a Test Control Table to control the processing load and test duration. Each DINS test extended over a period of six seconds and the test was divided into one second "phases" by a software "clock."

Because of the similarity in the data gathering techniques, DINS used the same offline analysis program that was used for PINS.

Minor Cycle	Scheduled Tasks	Reads	Writes	Priv Writes	Signals	Waits	Schedules
0							
1	QP12,Q14,SP50,SP51, QP10	3		8			
2	SP40,SP41,SP42,SP43, SP44,SP45,SP46,SP47, SP4C	26	10(2*)				
3	SP52,SP53,SP54,SP35	6				1	
4	QP30,QP40,DP10,DP11	12	3	2			
5	SP55,SP58,SP36,SP59	3	3			1	
6	SP4B	7	3				
7	SP14				2		
8							
9	QP12, QP14		3				
10	SP40,SP41,SP42,SP43, SP44,SP45,SP46,SP47, SP4C,SP30,SP31,SP48, SP32,SP49,SP33	40	16(3*)			4	
11							
12	QP30,QP40,DP10,DP11	12	3	2			
13							
14	SP4B,QP31,QP41,DP12	19	12	1			
15	SP14						

* Of the total number of WRITES, the figure in ( ) indicates the number that result in asynchronous transmissions between processors.

Figure 3.2-5 DINS EXECUTIVE USAGE (IN NORMAL GPS-AIDED MODE) FOR THE FIRST SECOND OF A 6-SECOND CYCLE (Page 1 of 2)

Minor Cycle	Scheduled Tasks	Reads	Writes	Priv Writes	Signals	Waits	Schedules
16							
17	QP12,QP14			3			
18	SP40,SP41,SP42,SP43, SP44,SP45,SP46,SP47, SP4C	26	10(2*)				
19							
20	QP30,QP40,DP10,DP11	12	3	2			
21							
22	SP4B	7	3				
23	SP14				1		

The 1/16 sec. cycle outlined above (Minor Cycles 16-23) is repeated for the next twelve 1/16 sec. cycles (Minor Cycles 24-31, 32-39, ... 112-119).

120							
121	QP12,QP14			3			
122	SP40,SP41,SP42,SP43, SP44,SP45,SP46,SP47, SP4C	26	10(2*)				
123							
124	QP30,QP40,DP10,DP11	12	3	2			
125							
126	SP4B,CP50	7	3				8
127	SP14,CP10						

Figure 3.2-5 (Page 2 of 2)

Minor Cycle	Scheduled Tasks	Reads	Writes	Priv Writes	Signals	Waits	Schedules
0							
1	QP12,P14,SP50,SP51	3		3			
2	SP40,SP42,SP42,SP43, SP44,SP45,SP46,SP47, SP4C	26	10(2*)				
3							
4	QP30,QP40,DP10,DP11	12	3	2			
5							
6	SP4B	7	3				
7	SP14				1		
8							
9	QP12,QP14		3				
10	SP40,SP41,SP42,SP43, SP44,SP45,SP46,SP47, SP4C	26	10(2*)				
11							
12	QP30,QP409,DP10,DP11	12	3	2			
13							
14	SP4B	7	3				
15	SP14				1		

* Of the total number of WRITES, the figure in ( ) indicates the number that results in asynchronous transmissions between processors.

Figure 3.2-6 DINS EXECUTIVE USAGE (IN NORMAL GPS-AIDED MODE) FOR EACH OF THE LAST 5 SECONDS OF A 6-SECOND CYCLE
(Page 1 of 2)

Minor Cycle	Scheduled Tasks	Reads	Writes	Priv Writes	Signals	Waits	Schedules
The 1/16 sec. cycle outlined above (Minor Cycles 8-15) is repeated for the next thirteen 1/16 sec. cycles (Minor Cycles 16-23, 24-31,... 112-119).							
120							
121	QP12,QP14			3			
122	SP40,SP41,SP42,SP43, SP44,SP45,SP46,SP47, SP4C	26	10(2*)				
123							
124	QP30,QP40,DP10,DP11	12	3	2			
125							
126	SP4B,CP50	7	3				8
127	SAP14,CP10						

Figure 3.2-6 (Page 2 of 2)

4.0 Data Analysis Summary

The Phase I testing measured the DAIS executive performance in a controlled load situation. The output of Phase I, reported in paragraph 4.2, is a set of elapsed processing times which cover any individual executive action. Working from this known executive performance, a set of algorithms was developed for the master and local executives which allows a user to predict executive overhead whenever the operating environment is known. These algorithms are presented in paragraph 4.3. Phase II testing measured the performance of the DAIS executive while supporting a navigation system (DINS). The executive overhead for DINS was calculated using the predictive algorithms and these a priori numbers were compared to the actual measurements made on DINS. These results are presented in paragraph 4.4.

All measurements made on the DAIS executives were collected through the use of software "hooks." Although the software hooks in no way modified the execution sequence in the DAIS executives, they did require extra processing time. The procedure used to determine instrumentation overhead is presented in paragraph 4.1 below.

4.1 Instrumentation Overhead

Each measurement made on the DAIS executive requires two software hooks. An event is trapped by instrumenting one hook at the very beginning of the event sequence and a second hook at the end of the event sequence. The first hook records the time when the event starts and the second hook records the time when the event is completed. A measurable amount of CPU time is required to execute these software hooks and record the data. Since this CPU time will show up as increased overhead in the executives, all measurements must be corrected for the hook (instrumentation) overhead before being used in the predictive algorithms. All results given in this report are identified as either corrected or uncorrected for the instrumentation overhead.

4.1.1 Method of Calculation

Instrumentation overhead was predicted by applying the processor instruction set timing estimates to each instruction in a hook and summing the results. This method was applied to all hooks used in the study. The timing estimates are listed for each test in paragraph 4.2. These predicted numbers were validated in the laboratory by obtaining measured data on the instrumentation overhead. This was done in the following manner. First, one of the tests described in paragraph 4.2 (e.g., master interrupt service overhead) was chosen for the master executive. The test was instrumented and data was collected by using one of the Test Control Tables described in appendix A (e.g., TCT PINS3). The offline analysis program generated output resembling that shown in figure 3.1-6. From this output was extracted the number of hooks executed in each test phase and the amount of available processor time that was used by the master executive. The instrumentation test was then run a second time, only for this run all of the hooks were eliminated from the master executive. Again, the offline analysis program was used to find the amount of CPU time required by the master executive. By dividing the difference in CPU time by the number of hooks, the average processor time required by a software hook in the master executive can be

determined. The entire procedure was repeated in an analogous manner to calculate the average time required for a software hook in the local executive.

4.1.2 Data and Results

A sample of the data obtained from the instrumentation overhead runs is shown in table 4.1-1. A series of such runs yielded the following average hook times in the DAIS executives:

Average hook time in master executive - 43 us
Average hook time in local executive - 50 us

	Processing Time				
Test Phase	With Hooks	Without Hooks	Hook Overhead	Number of Hooks	Processing Time per Hook
1	446 ms	409 ms	37 ms	844	43.8 us
2	509 ms	468 ms	41 ms	966	42.4 us
3	510 ms	468 ms	42 ms	972	43.2 us
4	502 ms	462 ms	40 ms	972	41.2 us
5	501 ms	461 ms	40 ms	974	41.1 us
6	507 ms	466 ms	41 ms	972	42.2 us
7	510 ms	468 ms	42 ms	966	43.5 us
8	508 ms	467 ms	41 ms	976	42.0 us
9	507 ms	467 ms	40 ms	970	41.2 us
10	508 ms	466 ms	42 ms	972	43.2 us

Executive Instrumented - Master
Test Performed - Interrupt Service Overhead
Test Control Table - PINS3
Special Conditions - Applications Tasks Disabled in Master Processor

Table 4.1-1 Calculation of Instrumentation Overhead - Sample Data

It is instructive to compare these measured times to the predicted values derived from the instruction timing, provided in paragraphs 4.2. As an example, the predicted average hook time for the master executive Interrupt Service Overhead test is:

$$T_m = (PHT_s + PHT_e)/2;$$

where:

T_m = Predicted average hook time, master executive
PHT_s = Predicted start hook time
PHT_e = Predicted end hook time

The predicted average hook time for the local Interrupt Service Overhead test (assuming the number of interrupts 3 matches the number of interrupts 5) is:

$$T_r = (PI3_s + PI3_e + PI5_s + PI5_e)/4;$$

where:

T_r = Predicted average hook time, remote
$PI3_s$ = Predicted start hook time for Interrupt 3
$PI3_e$ = Predicted end hook time for Interrupt 3
$PI5_s$ = Predicted start hook time for Interrupt 5
$PI5_e$ = Predicted end hook time for Interrupt 5

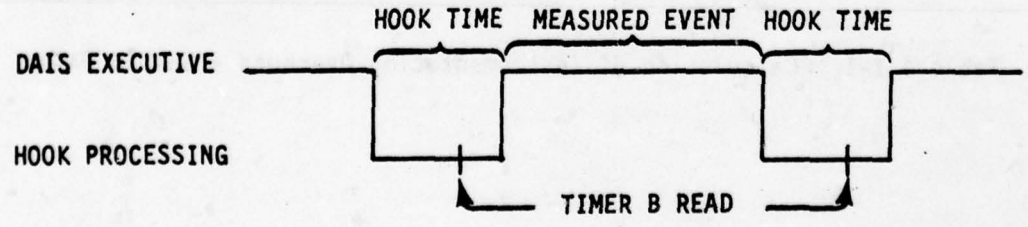
Inserting the predicted values derived from the instruction timing given in paragraph 4.2.2.2.

T_m = (47.4 us + 41.2 us)/2 = 44.3 us
T_r = (45.8 us + 48.8 us + 48.8 us + 48.8 us)/4 = 48.0 us

These predicted values are in excellent agreement with the measured values of 43 us and 50 us, respectively.

4.1.3 Correcting Test Results for Instrumentation Overhead

Each of the measurements made on the DAIS executives include the time required to process one software hook. This is illustrated on the following time line:



The time measured for the event is the difference in the two timer B values recorded. The interval between the two timer B values includes the second portion of the first hook and the first portion of the second hook, or the equivalent of one hook time.

All of the measured times provided in paragraphs 4.2 and 4.4 are specifically identified as being either corrected for or uncorrected for the instrumentation overhead.

4.2 DAIS Executive Evalution Test Descriptions

Phase I of the DAIS Executive Evaluation comprised six separate tests: Transmission Delay Time (TDT), Interrupt Service Overhead (ISO), System Response Time (SRT), Event Service Overhead (ESO), Master Executive Overhead (MEO), and Local Executive Overhead (LEO). Each test is described individually in the paragraphs that follow. The information provided for each test includes:

(a) Test Instrumentation - A description of the software hooks used and the locations chosen for them. For each test, a time line is developed which shows the sequence of events and describes the factors that could influence the service time at critical points.

(b) Instrumentation Timing - Timing estimates for each of the hooks used in the test.

(c) Test Control Tables - A list of the test control tables used for the test.

(d) Performance Data - A summary of the test data collected. For each test, sufficient data samples were obtained to produce average values accurate to 1% or better. For measurements in the 100 us range, thousands of test samples were collected to provide this accuracy.

4.2.1 Test #1 - Transmission Delay Time

Transmission Delay Time (TDT) is defined as the delta time from a ready-to-transmit state in the DAIS executive to the time when the bus transmission has been completed. See the "Test Plan for the DAIS Executive Evaluation Program" for a more detailed discussion of the transmission delay time test.

A SCHEDULE statement requiring an Interprocessor Service Request was selected as the "transmission" used to measure TDT. This choice was made because the SCHEDULE generates a high-priority request which is easy to identify during realtime.

4.2.1.1 TDT Test Description

The TDT test is only instrumented in the remote processor. Since a SCHEDULE is used as the message, the local executive routine that queues the request (X$IPSR) is instrumented and the routine which dequeues the request (X$ATRO) upon transmission is instrumented.

4.2.1.1.1 Instrumented Routines

X$IPSR Instrumentation

Routine X$IPSR is instrumented with a JS (jump to subroutine) instruction at an offset of $1E_{16}$ (X'1E') into the routine. This location was chosen because the Asynchronous ID has been stored in a static location in routine X$IPSR and is readily available. The instruction at this location, which is overstored with the jump to the instrumentation subroutine, is saved in the Executive Evaluation Software (EES) module.

Location	Contents	Description of Contents
*M X$IPSR + X'1E'**	7220	JS 2,
M X$PISR + X'1F'	5000	TDTR1
M TDTSV1	8030	(Overstored instruction -
M TDTSV1+1	048C	now stored in TDTSV1 of the EES module)

X$ATRO Instrumentation

Routine X$ATRO is instrumented with a JS instruction at offset X'8'. This location was chosen since a transmission has been completed and the asynchronous ID is available for comparison with the desired ID. The instruction at this location which is overstored with the jump to subroutine instruction is saved in the EES module.

Location	Contents	Description of Contents
*M X$ATRO + X'08'**	7230	JS 3,
M X$ATRO + X'09'	5012	TDTR2
M TDTSV2	8021	Overstored instruction -
M TDTSV2+1	028C	now stored in TDTSV2 of the EES module

4.2.1.1.2 Factors Affecting Transmission Delay Time

Sequence of Events:

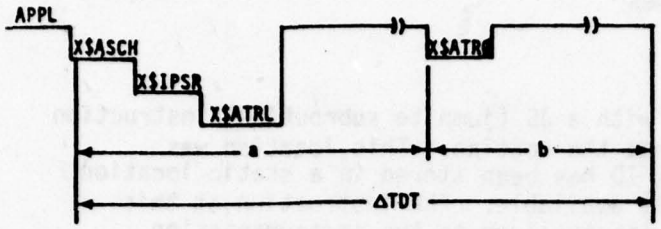
1. Applications task requests an executive service (SCHEDULE) which requires an interprocessor service request.

2. An interprocessor service request is queued in the asynchronous transmission queue.

3. When the remote processor next transmits a status word on the bus, a status exception is raised.

4. The master executive responds to the status exception and effects the transmission.

* "M" signifies a modification of the referenced location.
** "X" refers to a hexidecimal offset.



Factors Affecting the Length of the Time Line:

Point (a) in the time line is dependent upon the number of asynchronous transmissions already in the asynchronous transmission queue. Previously queued asynchronous requests must be handled first.

Point (b) in the time line is dependent upon the master executive responding to the status exception and effecting this transmission which is determined by what the master must do prior to soliciting a status response from the remote processor (i.e., synchronous bus list, polling list, etc.)

4.2.1.2 TDT Instrumentation Timing

These estimates are based upon the instruction times provided in "DAIS Processor Instruction Set," SA 401301.

X$IPSR	(with untrue compare on ASYNCH ID)	20.6 usec
X$IPSR	(with true compare on ASYNCH ID)	56.2 usec
X$ATRO	(with untrue compare on ASYNCH ID)	21.2 usec
X$ATRO	(with true compare on ASYNCH ID)	56.8 usec

4.2.1.3 TDT Test Control Tables

Test Control Tables SACIA and PINS2 were used to study TDT. A description of these test control tables can be found in paragraphs A.2.1 and A.2.2 of appendix A.

4.2.1.4 TDT Performance Data

Transmission Delay Time (TDT) was measured by adding the SCHEDULE statement used to find TDT to the base load shown in figure 3.1-5. The SCHEDULE statement was added to each minor cycle in turn to obtain averaged data for TDT. This data is shown in table 4.2-1.

Minor cycles 1, 2, 3, 4, and 7 show very similar results for Transmission Delay Time. The average TDT for these five minor cycles will be referred to as the "nominal" TDT. Minor cycles 0, 5, and 6 show substantial increases over the nominal TDT. The reason for these differences can be found by studying figure 3.1-5. Minor cycles 0, 5, and 6 are the only ones in the base load that have bus activity. BCIU transmissions of synchronous COMPOOL blocks occur in minor cycles 0 and 5, while minor cycle 6 has an asynchronous transmission of a COMPOOL block.

Minor Cycle Number	Measured	Corrected
Minor Cycle 0	1919 us	1869 us
Minor Cycle 1	1375 us	1325 us
Minor Cycle 2	1350 us	1300 us
Minor Cycle 3	1388 us	1338 us
Minor Cycle 4	1450 us	1400 us
Minor Cycle 5	2631 us	2581 us
Minor Cycle 6	3088 us	3038 us
Minor Cycle 7	1463 us	1413 us

Table 4.2-1 Transmission Delay Time Data

TDT increases when BCIU transmissions occur because the master processor must respond to a status exception raised in the remote processor. The master processor is unaware of the status exception until a status word is received from the remote processor. If BCIU transmissions are in progress, idle polling is disabled so there is no regular transmission of status words to the master processor. The status word cannot be generated until either idle polling is enabled again or a BCIU operation involving the remote processor causes a status word to be generated. It is this delay caused by a "busy" bus which lengthens the TDT in minor cycles 0 and 5.

The above-nominal TDT in minor cycle 6 has a different cause. In this minor cycle, an applications task performs an asynchronous WRITE of a global COMPOOL block. This request is placed in the asynchronous transmission queue before the SCHEDULE statement is executed. Thus the SCHEDULE request cannot be sent to the master processor until the global WRITE is completed, causing the observed increase in TDT.

TDT will always be above nominal if there is already an entry in the asynchronous transmission queue when the statement of interest is executed. This was verified by running TDT tests with TCT SACIA. As just one example of the manner in which TDT changed when another asynchronous bus transmission existed, the addition of an asynchronous global WRITE to minor cycle 1 caused the measured TDT to increase from 1375 us to 3268 us.

The most useful estimate of TDT is for optimum (or minimum) response. The minimum TDT occurs when idle polling is in effect on the bus and the asynchronous transmission queue is empty. This will be the prevailing situation in a program designed to DAIS standards. Good system design will entail infrequent use of asynchronous transmissions (so the queue will generally be empty) and heavy use of the bus will be confined to minor cycles devoid of applications tasks that might require bus support. TDT measurements were obtained for five minor cycles which had no other bus activity. Averaging these measurements gives a TDT of 1405 us. Correcting this value for the instrumentation overhead produces the following result:

Optimum Transmission Delay Time = 1355 us.

In the following paragraph TDT estimates are generated for situations other than the optimal one.

4.2.1.5 TDT Theoretical Modeling

In this section we develop a mathematical model for the prediction of the average asynchronous Transmission Delay Time in more general circumstances. It is felt that since the times of occurance and times of service of asynchronous events are inherently random, a statistical model is necessary for complete analysis. This model is formed using known results from statistical queuing theory and measured DAIS parameters.

For the purposes of this work, a mathematical model is used in which several assumptions must be made. These assumptions are only partially justified at this time since the model is used for predictive purposes only and should be judged solely on that basis. From this model we can show, theoretically, the effects of system utilization on asynchronous transmissions. We then check our model by comparing theoretical prediction with the data presented in section 4.2.1.4.

We begin by describing the model we use for this work. It is assumed that the system (composed of asynchronous transmission requests, the bus transmission queue and bus transmissions) behaves like a single server queue with a Markov interarrival time distribution and a Markov service time distribution. That is, we assume that asynchronous transmission requests are distributed randomly in the applications program and the time between requests has an exponential distribution (described below). Furthermore, we assume that the time necessary to service each asynchronous transmission request is a random variable with an exponential distribution. Random service times are justified by assuming the asynchronous activity of remote terminals ties-up the bus for random periods of time as discussed under "Factors Affecting the Length of the Time Line" in paragraph 4.2.1.1. We quantify the above by letting:

$\tilde{t}$ = time between asynchronous transmission requests (a random variable) and

$\tilde{x}$ = service time, that is, time required to perform the transmission once the request reaches the head of the queue (also a random variable)

Defining

$F_{\tilde{t}}(t) = P(\tilde{t} \leq t)$ (the probability that $\tilde{t}$ is less than or equal to t), and
$F_{\tilde{x}}(x) = P(\tilde{x} \leq x)$ (the probability that $\tilde{x}$ is less than or equal to x)

then by exponential waiting times we mean

$$f_{\tilde{t}}(t) = \frac{d}{dt} F_{\tilde{t}}(t) = \lambda e^{-\lambda t} \qquad (t \geq 0), \text{ and}$$

$$f_{\tilde{x}}(x) = \frac{d}{dt} F_{\tilde{x}}(x) = \mu e^{-\mu x} \qquad (x \geq 0)$$

(Both densities are zero for arguments less than zero.)

By specifying λ, the average number of asynchronous requests per second and μ, the average number of transmissions per second, we completely specify the statistical behavior of the queue. Note that

$E(\tilde{t}) = \overline{t} = 1/$ (average interarrival time), and
$E(\tilde{x}) = \overline{x} = 1/u$ (average service time)

where E represents mathematical expectation (the "averaging operator").

We define Transmission Delay Time as the time spent waiting in the transmission queue (waiting time) plus service time. Hence, average transmission delay time is given by

$$\overline{TDT} = \overline{x} + W$$

where W is the average waiting time. Our model will predict $\overline{TDT}$. In this model the key parameter is the system utilization factor, ρ. We define ρ by: $\rho = \lambda \overline{x}$; it may be thought of as the ratio of the rate at which asynchronous transmission requests arrive to the capacity of the system to transmit these messages. It may also be thought of as the average fraction of the available time that transmissions take place.

The average number of requests per second is determined by the applications programs. The average service time is dependent on the length of time taken to transmit the message and, in our model, it is also dependent upon the amount of bus activity other than the asynchronous activity requested by the master processor. That is, we consider the delays caused by all synchronous bus activity and the asynchronous activity not requested by the local processor as additional average service time, $\overline{x}$.

For our particular queue, we find from Kleinrock (Queueing Systems, Volume II; Computer Applications, Wiley Interscience, (New York), 1976) that:

$$W = \frac{\rho/\mu}{1-\rho}$$

and hence,

$$\overline{TDT} = \frac{1/\mu}{1-\rho}$$

From the data analysis in the previous section, we find that the average transmission time (service time) is approximately $\overline{x} \approx 1355$ usec. Using this value, we obtain the plot shown in figure 4.2-1. For average Transmission Delay Time ($\overline{TDT}$) vs ρ.

In order to finish (on the average) the transmission load placed on the system in one minor cycle before the following minor cycle begins loading the system, we require that the average transmission delay time be less than the minor cycle period, 7812.5 usec. From figure 4.2.1, this corresponds to $\rho = 0.8266$ which in turn corresponds to

$$\lambda = \frac{\rho}{\bar{x}} = \frac{0.8266}{1355 \times 10^{-6}} = 610 \text{ events/sec}$$

that is, 610/128 = 4.77 events/minor cycle. Hence, we should expect to perform no more than 4.77 asynchronous transmissions per minor cycle on the average without falling behind. Indeed, the data collected thus far indicates that no more than four asynchronous transmissions per minor cycle are accomplished (by the remote processor), and on the average slightly less than four per minor cycle are observed. Taking into account the small number of minor cycles sampled (8), as well as the unverified modeling assumptions specified earlier, the agreement is considered reasonable.

For minor cycles 0, 5, and 6 (refer to figure 3.1-5) we make the assumption that the additional bus activity is uniformly distributed over a minor cycle and that this may be viewed as additional service time. For example, from table 4.2-1, we use an average service time ($\bar{x}$) of 1869 usec for minor cycle 0 and an average service time of 2581 usec for minor cycle 5. The entry for minor cycle 6 represents the average service time for two asynchronous transmissions and hence the average service time for one transmission is 1519 usec. When we apply the above analysis to minor cycles 0, 5, and 6 we obtain the curves shown in figure 4.2-2 for $\overline{TDT}$ vs ρ. Continuing the analysis, we develop the table below which gives the expected and the observed maximum number of asynchronous transmissions per minor cycle based on the assumption that the maximum $\overline{TDT}$ allowable is the length of one minor cycle.

Minor Cycle Number	Theoretical Maximum Average Number of Transmissions	Observed Maximum Average Number of Transmissions
0	3.18	3.1
5	2.03	2.0
6	4.14	3.5
1, 2, 3, 4, 7	4.73	3.5

This table may be considered as a check of the theory developed in this section and as a verification of the assumptions behind the theory. It appears that the model is accurate for very busy periods and is somewhat optimistic for periods with little or no additional bus activity. Once again, the small number of minor cycles sampled must be considered when assessing the results.

The primary conclusion that can be drawn from this analysis is the unsurprising statement that this executive is amenable to processing only a few asynchronous transmissions per minor cycle.

4.2.2 Test #2 - Interrupt Service Overhead

Interrupt Service Overhead (ISO) is defined as the time from the receipt of an interrupt signal to the time of the return to the interrupted program (either an actual return or a transfer of control to some controlling program). See the "Test Plan for the DAIS Executive Evaluation" for a more detailed discussion of the Interrupt Service Overhead test.

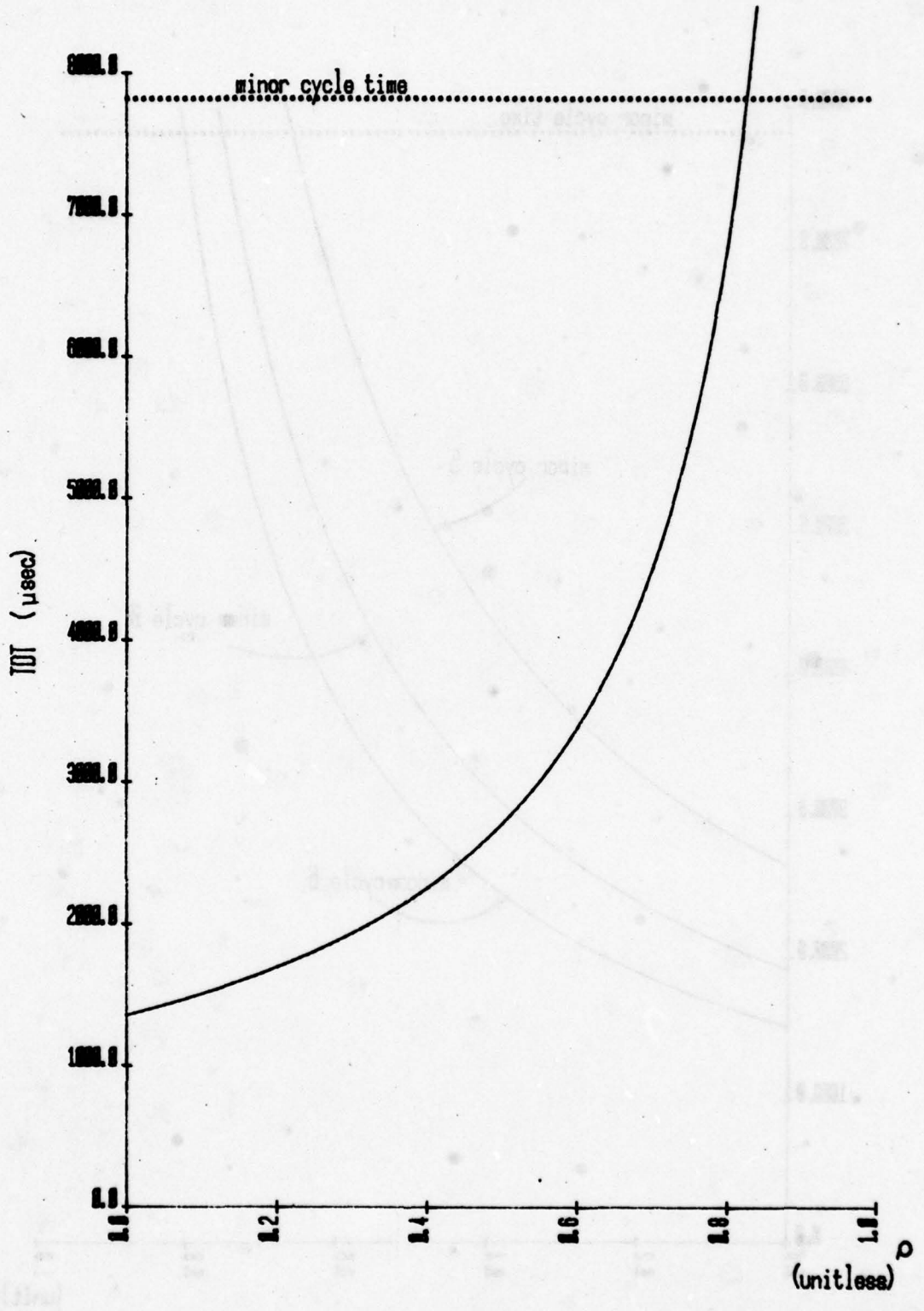


Figure 4.2-1 Average Transmission Delay Time vs. System Utilization Factor

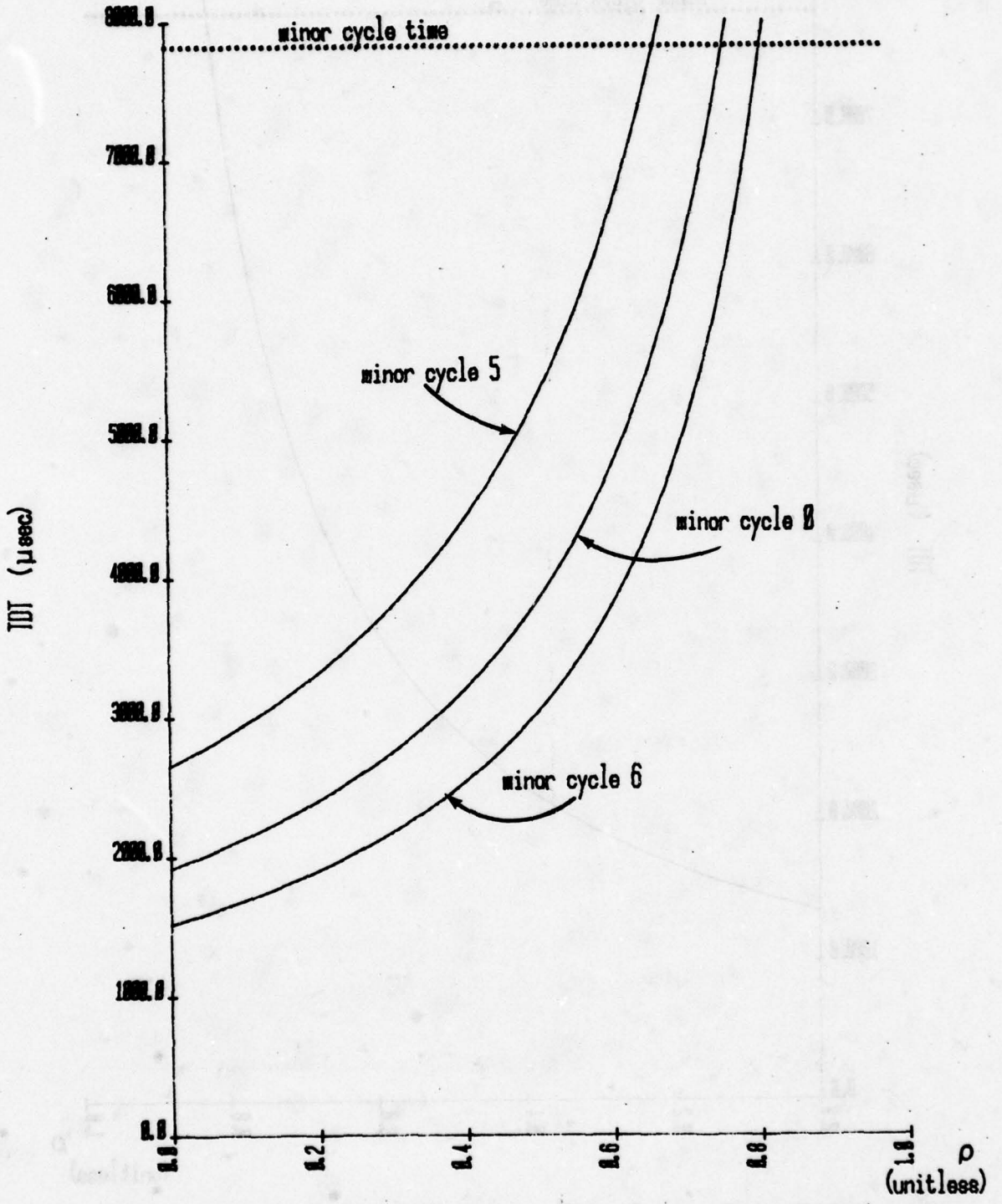


Figure 4.2-2 Average Transmission Delay Time vs. System Utilization Factor with Additional Bus Loading

4.2.2.1 ISO Test Description

The Interrupt Service Overhead test is instrumented in both the master and remote processors since each processor responds to a different set of interrupts.

4.2.2.1.1 ISO Master Processor Test Description

4.2.2.1.1.1 Instrumented Routine

In the master processor all interrupts are fielded through one common routine, M$INTH. The address of the appropriate interrupt handling routine is loaded into register 1 and a jump to subroutine is executed to cause the transfer of control. The return from the interrrupt handling routine is always through the address in the linkage register 2. The jump to subroutine via register 1 instruction is replaced by a jump to the interrupt service overhead data gathering routine.

M$INTH Instrumentation:

Location	Contents	Description of Contents
*M M$INTH + X'C2'**	7220	JS 2,
M M$INTH + X'C3'	7F00	ISOM1

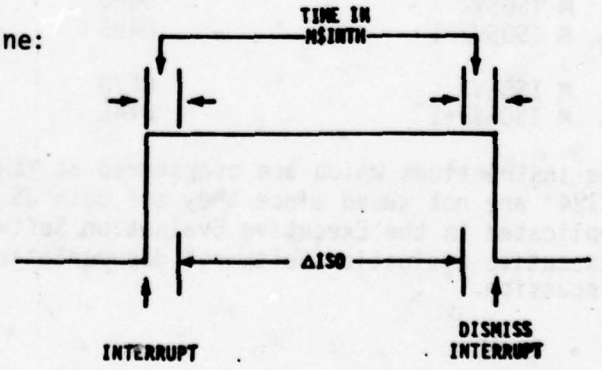
Only one instrumentation hook is required since routine ISOM1 ensures that upon completion of the interrupt service, ISOM2 will be entered to record the end of interrupt service. See "Executive Evalution Software" program documentation for a more detailed description of these subroutines.

4.2.2.1.1.2 Factors Affecting Interrupt Service Overhead - ISO (Master)

Sequence of Events:

1. Hardware interrupt is taken and routine M$INTH is entered.
2. M$INTH calls M$BINT (for bus interrupt), M$GINT (for general interrupt) or M$TIMA (for timer A interrupt).
3. Interrupt is handled and return is to M$INTH to dismiss the interrupt.

4.2.2.1.1.3 Master ISO Time Line:



*"M" signifies a modification of the referenced location
**"X" indicates an offset

4.2.2.1.2 ISO Remote Processor Test Description

4.2.2.1.2.1 Instrumented Routines

In the remote processor all interrupts are fielded through one common routine, X$REM1 and the actual handling of the interrupt is through X$AREC, X$ATRO and a small internal subroutine in X$REM1 named MC. After interrupt service, return is always through internal subroutines SUSPEN or RETN in X$REM1.

X$REM1 Instrumentation

Location	Contents	Description of Contents
M X$REM1 + X'FC'	7220	JS 2,
M X$REM1 + X'FD'	5026	ISOR1
M X$REM1 + X'11E'	7220	JS 2,
M X$REM1 + X'11F'	504C	ISOR4
M X$REM1 + X'124'	7220	JS 2,
M X$REM1 + X'125'	5032	ISOR2
M X$REM1 + X'126'	70F0	JC 15,
M X$REM1 + X'127'	505A	ISOR5
M X$REM1 + X'130'	7220	JS 2,
M X$REM1 + X'131'	503E	ISOR3

The instructions which are overstored at X$REM1 + X'11E', X$REM1 + X'124' and X$REM1 + X'130' are saved in the Executive Evaluation Software (EES) module:

Location	Contents	Description of Contents
M ISOSV1	8A00	Overstored instruction
M ISOSV1+1	8008	from X$REM1 + X'130'
M ISOSV2	50F0	Overstored instruction
M ISOSV2+1	0495	from X$REM1 + X'11E'
M ISOSV3	8F20	Overstored instruction
M ISOSV3+1	014E	from X$REM1 + X'126'

The instructions which are overstored at X$REM1 + X'FC' and X$REM1 + X'124' are not saved since they are both JS 2 instructions which are replicated in the Executive Evaluation Software module. See the "Executive Evaluation Software" documentation for a more detailed discussion.

4.2.2.1.2.2 Factors Affecting Interrupt Service Overhead - ISO (Remote)

Sequence of Events:

1. Hardware interrupt is taken and routine X$REM is entered.
2. X$REM calls X$AREC or X$ATRO (for bus interrupts) or sets the minor cycle.
3. X$REM dismisses the interrupt.

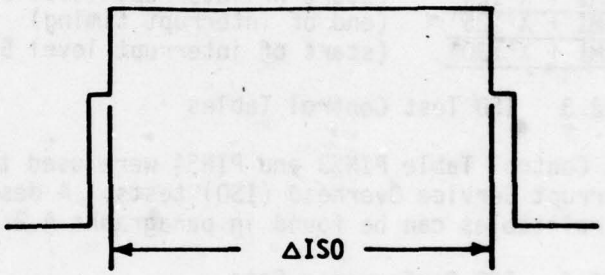
4.2.2.1.2.3 Remote ISO Time Line:

Interrupt 3

X$AREC/X$ATRO

Executive Interrupt Handling Routine

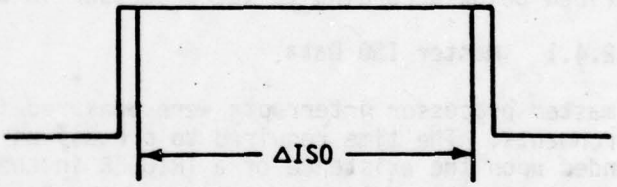
Interruptable Routine



Interrupt 5

Executive Interrupt Handling Routine

Interruptable Routine



4.2.2.1.2.4 Factors Affecting Length of Time Line

Interrupt 3:

The time spent in X$ATRO is affected by the presence of another message in the asynchronous transmission queue

The time spent in X$AREC depends upon whether this is a reception or a retransmit request

Interrupt 5:

The time line is fixed for an interrupt 5.

4.2.2.2 ISO Instrumentation Timing

These estimates are based upon the instruction times provided in "DAIS Processor Instruction Set," SA 401301.

4.2.2.2.1 ISO Master Processor Instrumentation Timing

M$INTH	(at start of ISO Timing - ISOM1)	47.4 usec
M$INTH	(at end of ISO Timing - ISOM2)	41.2 usec

4.2.2.2.2 ISO Remote Processor Instrumentation Timing

X$REM1 + X'FC'	(start of interrupt level 3 timing)	45.8 usec
X$REM1 + X'11E'	(end of interrupt timing)	48.8 usec
X$REM1 + X'124'	(start of interrupt level 3 timing)	45.8 usec
X$REM1 + X'126'	(end of interrupt timing)	48.8 usec
X$REM1 + X'130'	(start of interrupt level 5 timing)	48.8 usec

4.2.2.3 ISO Test Control Tables

Test Control Table PINS3 and PINS4 were used to generate data for the Interrupt Service Overhead (ISO) tests. A description of these test control tables can be found in paragraphs A.2.3 and A.2.4 of appendix A.

4.2.2.4 ISO Performance Data

The time required by the DAIS executive to process all major interrupts was measured in this test. Certain interrupts which occur infrequently were not measured (e.g., the interrupt signalling a rollover in timer B occurs every 6.5 seconds and was not measured). The interrupts are described below according to the processor in which they occur.

4.2.2.4.1 Master ISO Data

All master processor interrupts were measured in a variety of operating environments. The time required to process an interrupt 10 (timer A) depended upon the existence of a TRIGGER in the system. The major interrupts in the master processor and the time taken to process them are:

Interrupt	Description	Measured Time	Corrected Time
1	BCIU Halt	219 us	176 us
3	Asynchronous completion (either reception or transmission)	281 us	238 us
5	Status exception (asynchronous request)	398 us	355 us
10	Timer A (minor cycle completion with no TRIGGER queued)	226 us	183 us
10	Timer A (minor cycle completion with a TRIGGER queued)	362 us	319 us
10	Timer A (upon completion of TRIGGER)	190 us	147 us

Table 4.2-2 Master Processor Interrupt Times

The measured times include the instrumentation overhead. After correcting the measured values for the instrumentation overhead, the true interrupt times in the master processor are shown in the last column of table 4.2-2.

Figure 4.2-3 depicts the overhead time used for master processor interrupts as a function of the number of interrupts encountered for interrupts 1, 3, 5, and 10 (TRIGGER completed). The straight lines indicate that the overhead time per interrupt does not vary as a function of processor load.

4.2.2.4.2 Remote ISO Data

All remote interrupts were measured in a variety of operating environments. The measured interrupt times do not vary with processing load; however, interrupt 3 times depend upon whether the completed BCIU process was a reception or a transmission. The measured times for the remote interrupts are:

Interrupt	Description	Measured Time	Corrected Time
3	Asynchronous Transmission	127 us	77 us
3	Asynchronous Reception	196 us	146 us
5	Minor Cycle Mode Code Receipt	114 us	64 us

Table 4.2-3 Remote Processor Interrupt Times

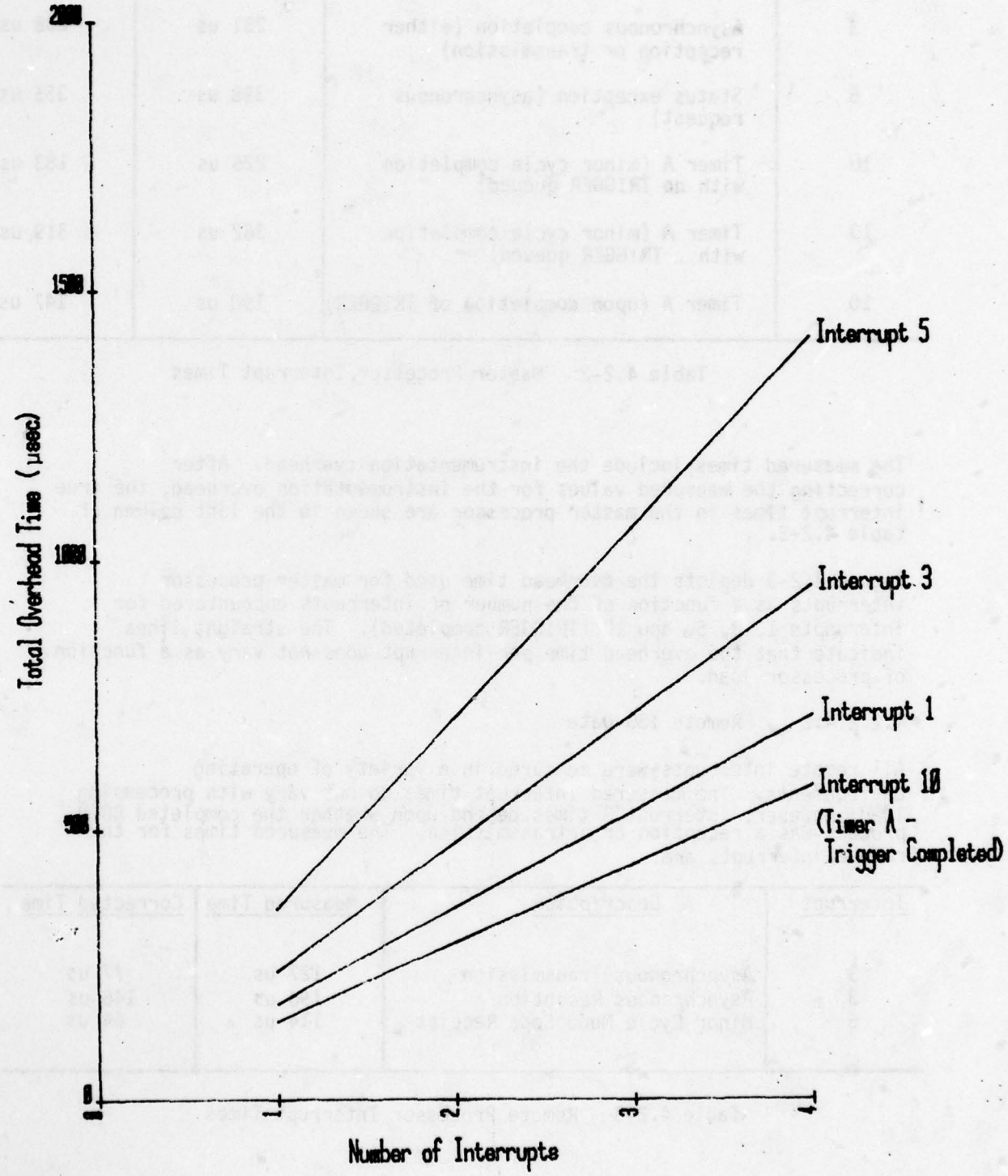


Figure 4.2-3 Master Processor Interrupt Overhead Times vs. Number of Interrupts

The measured times include the instrumentation overhead. These measurements were corrected for the overhead incurred, and the corrected interrupt times for the remote processor are shown in the last column of table 4.2-3.

Figure 4.2.4 depicts the overhead time used for the remote processor as a function of the number of interrupts 3 encountered (interrupt 5 occurs only once per minor cycle and is not shown). The straight lines indicate that the overhead time per interrupt does not vary as a function of processor load.

4.2.3 Test #3 - System Response Time

System Response Time (SRT) is defined as the amount of time between a ready-to-transmit state in the DAIS executive and receipt of a response to that transmission. For this particular measurement, an interprocessor service request is used for both the transmission and the response.

To allow measurement of the System Response Time, a special PINS task resides in each processor. The first task requests a SCHEDULE of the second task (which is resident in the other processor) and the first task then requests a WAIT for event. When the second task is activated, it will SIGNAL the event to the first task.

4.2.3.1 SRT Test Description

The System Response Time test is instrumented in both processors and is the same in both. Since a SCHEDULE realtime statement is used to start the timing, the routine which queues the schedule request, X$IPSR, is instrumented. Also, a SIGNAL from the other processor will end the System Response Time timing so the event handling routine, X$EVHA, is instrumented.

4.2.3.1.1 SRT Master Processor Test Description

4.2.3.1.1.1 Instrumented Routines

The start of timing is in the interprocessor service request routine X$IPSR.

X$IPSR Instrumentation

Location	Contents	Description of Contents
*M X$IPSR + X'1E'**	7220	JS 2,
M X$IPSR + X'1F'	7F16	SRTM1

The instruction which is overstored at X$IPSR + X'1E' is saved in the executive evaluation software for execution prior to return to the inline code of X$IPSR.

*"M" signifies a modification of the referenced location.
**"X" signifies an offset.

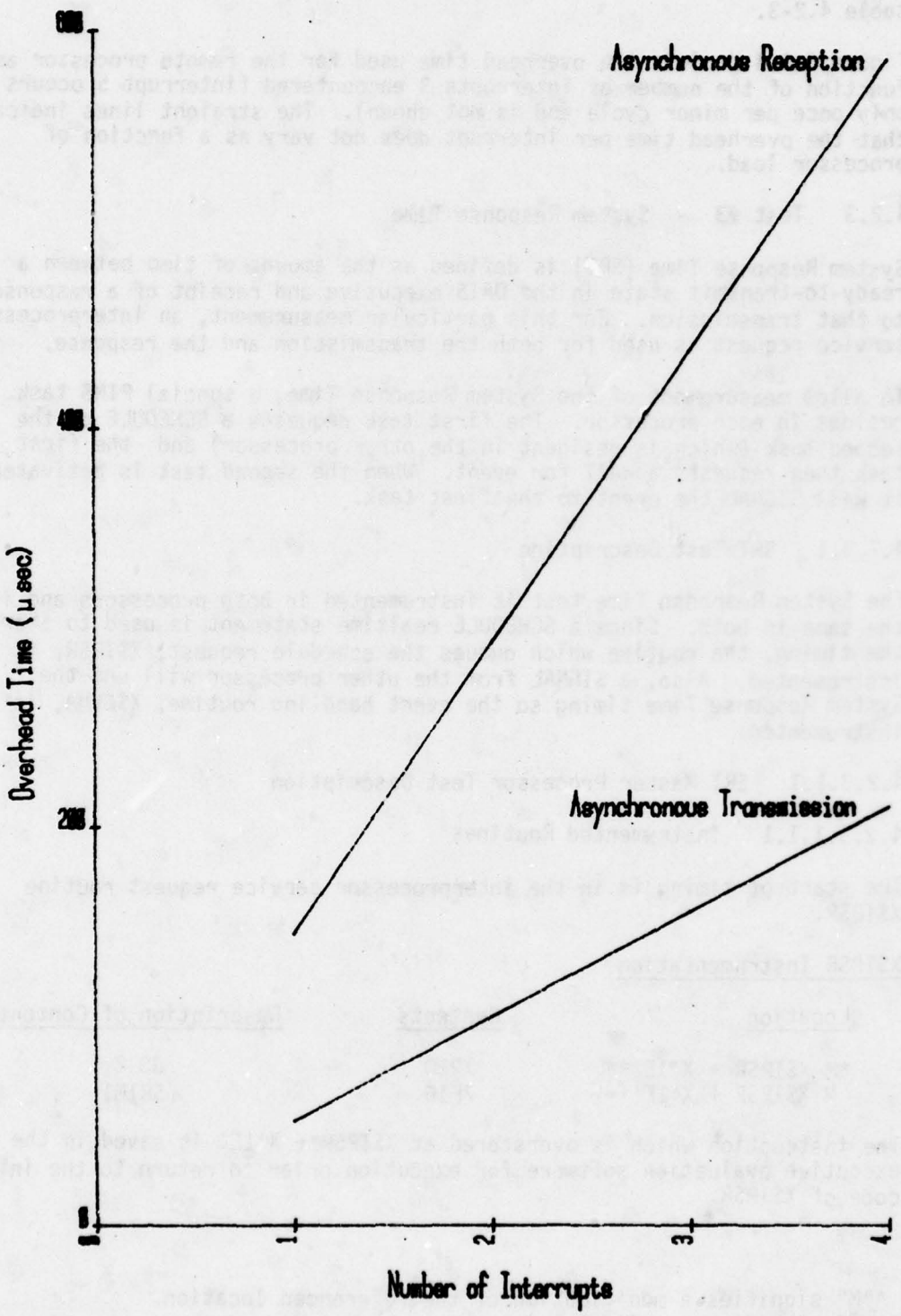


Figure 4.2-4 Remote Processor Interrupt 3 Overhead Times vs. Number of Interrupts

Location	Contents	Description of Contents
M SRTSV1	8030	Overstored instruction
M SRTSV1+1	048C	from X$IPSR + '1E'

The end of timing is in the event handling routine X$EVHA.

X$EVHA Instrumentation

Location	Contents	Description of Contents
M X$EVHA + X'8'	7220	JS 2,
M X$EVHA + X'9'	7F28	SRTM2

The instruction which is overstored at X$EVHA + X'8' is saved in the executive evaluation software for execution prior to return to the inline code of X$EVHA.

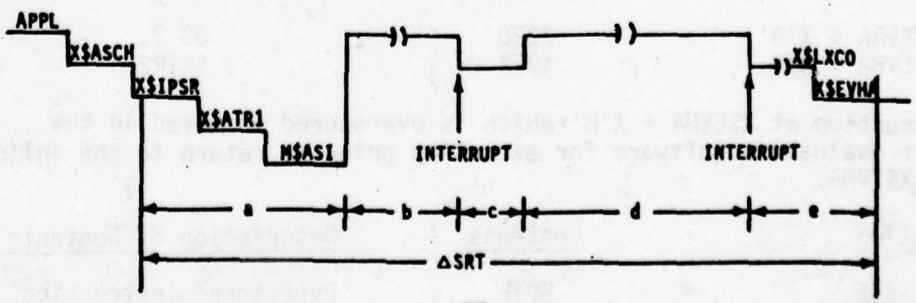
Location	Contents	Description of Contents
M SRTSV2	801F	Overstored instruction
M SRTSV2+1	0001	from X$EVHA + 8

4.2.3.1.1.2 Factors Affecting System Response Time - SRT (Master)

Sequence of Events:

1. Applications task in master processor issues SCHEDULE request for counterpart task in remote processor.
2. *Interprocessor Service Request program* queues SCHEDULE request for other task.
3. Applications task enters WAIT for event SIGNAL from counterpart task.
4. Schedule request is transmitted to remote processor.
5. Task is SCHEDULEd in remote processor and is activated.
6. Task in remote processor SIGNALs event which is queued as an interprocessor request and a status exception is raised.
7. Master responds to the status exception and asynchronous transmission of the request is started.
8. SIGNAL is queued and local executive in master processor is notified.

4.2.3.1.1.3 Master SRT Time Line



4.2.3.1.1.4 Factors Affecting Length of Time Line

Items a - e below refer to segments of the above time line.

a. if bus is not active, X$ATR1 and M$ASI are called; otherwise X$IPSR simply adds the request to the asynchronous transmission queue.

b. varies by the number of asynchronous transmissions in the queue ahead of the one being measured and a currently active synchronous transmission, if any. The interrupt that terminates time b is the interrupt that allows this request to be transmitted.

c. is fixed and is the time to effect this transmission when it reaches the top of the queue.

d. varies by the amount of time it takes the master processor to acknowledge the request for an interprocessor service request (SIGNAL) and ready it for transmission.

e. varies by master executive functions or local executive functions already pending when the SIGNAL is received.

4.2.3.1.2 SRT Remote Processor Test Description

The start of timing for the System Response Time test is in the Interprocessor Service Request routine X$IPSR.

X$IPSR Instrumentation

Location	Contents	Description of Contents
M X$IPSR + X'1E'	7220	JS 2,
M X$IPSR + X'1F'	5068	SRTR1

The instruction which is overstored at X$IPSR + X'1E' is saved in the executive evaluation software for execution prior to return to the inline code of X$IPSR.

Location	Contents	Description of Contents
M SRTSV1	8030	Overstored instruction
M SRTSV1+1	048C	from X$IPSR + X'1E'

The end of timing is in the event handling routine X$EVHA.

Location	Contents	Description of Contents
M X$EVHA + X'8'	7220	JS 2,
M X$EVHA + X'9'	407A	SRTR2

The instruction at X$EVHA + X'8' which is overstored is saved in the executive evaluation software for execution prior to return to the inline code of X$EVHA.

Location	Contents	Description of Contents
M SRTSV2	801F	Overstored instruction
M SRTSV2+1	0001	from X$EVHA + X'8'

4.2.3.1.2.2 Factors Affecting System Response Time - SRT (Remote)

Sequence of Events:

1. Applications task in remote processor requests SCHEDULing of counterpart task in master processor.

2. X$IPSR queues the request; the applications task enters a WAIT state.

3. Request becomes first in queue and status exception is raised.

4. Master responds to status exception and effects asynchronous transmission.

5. Local executive in master schedules and activates counterpart task in master processor.

6. Counterpart task requests an interprocessor SIGNAL.

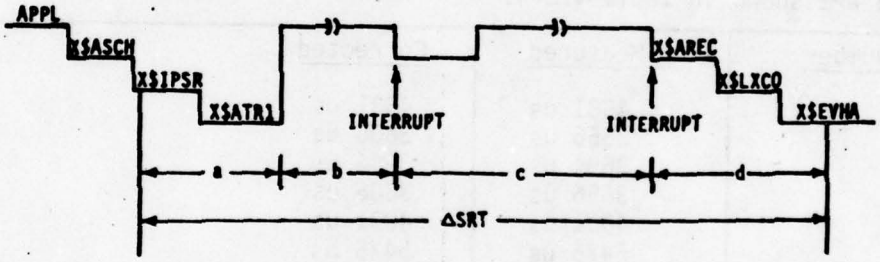
7. Interprocessor signal request is queued.

8. Interprocessor signal request becomes first in queue and master effects the transmission.

9. Interprocessor SIGNAL is received in remote processor.

10. Event handling routine completes application task WAIT.

4.2.3.1.2.3 Remote SRT Time Line



4.2.3.1.2.4 Factors Affecting Length of Time Line

Items a-d below refer to segments of the above time line.

a. if transmission queue is empty, X$ATR1 is called to raise a status exception.

b. is terminated by the interrupt which allows this request to be the next transmission. This time is affected by the bus list and the polling sequence which determines when the master will see the status exception.

c. is terminated by the interrupt resulting from the receipt of the asynchronous signal which the applications task is WAITing on. This time is dependent upon the number of asynchronous transmissions in the master processor asynchronous transmission.

d. this time is dependent upon the number of asynchronous receptions in the queue ahead of the SIGNAL request from the master.

4.2.3.2 SRT Instrumentation Timing

These estimates are based upon the instruction times provided in "DAIS Processor Instruction Set," SA 401301.

X$IPSR	(at start of SRT Timing)	
	(on true compare on ASYNCH ID)	55.0 usec
	(on untrue compare on ASYNCH ID)	21.4 usec
X$EVHA	(at end of SRT Timing)	
	(on true compare on EV'TAB'OFFS)	55.0 usec
	(on untrue compare on EV'TAB'OFFS)	21.4 usec

4.2.3.3 SRT Test Control Tables

Test control tables SACIA and PINS3 were used to measure the SRT. A description of these test control tables can be found in paragraphs A.2.1 and A.2.3 of appendix A.

4.2.3.4 SRT Performance Data

System Response Time was measured by adding the SCHEDULE statement used to measure SRT to the base load shown in figure 3.1-5. The SCHEDULE statement was added to each minor cycle in turn (omitting 7 since the load in 7 is identical to the load in 3) to obtain data for SRT. The averaged data are shown in table 4.2-4.

Minor Cycle Number	Measured	Corrected
0	4581 us	4531 us
1	3656 us	3606 us
2	3694 us	3944 us
3	3656 us	3606 us
4	4081 us	4031 us
5	5475 us	5425 us
6	5369 us	5319 us

Table 4.2-4 System Response Time Data

As found with the transmission delay time (see paragraph 4.2.1.4) the times obtained in minor cycles 0, 5, and 6 are substantially higher than the others. Once again, the cause of these increased response times can be traced to the existence of bus traffic in minor cycles 0, 5, and 6. BCIU transmissions of synchronous COMPOOL blocks occur in minor cycle 0 and 5, while an asynchronous transmission of a COMPOOL block occurs in minor cycle 6. System response time is slowed in minor cycles 0 and 5 because the bus is busy with synchronous transmissions and status exceptions cannot be recognized by the master executive until the current bus transmission is completed. System response time is longer in minor cycle 6 because a previous asynchronous request is already queued in the asynchronous transmission queue when the SCHEDULE request is processed. The previous request must be handled before the SCHEDULE request can rise to the top of the queue.

As with TDT, SRT will always be above nominal if there is a previous entry in the asynchronous transmission queue. This was demonstrated by running SRT tests with TCT SACIA. As just one example of the manner in which SRT changed when another asynchronous bus transmission existed, the addition of a TRIGGER to minor cycle 6 caused SRT to increase from 5369 us to 6819 us.

The most useful estimate of SRT is for optimum (or minimum) response. The optimum SRT occurs when idle polling is active on the bus and the asynchronous transmission queues in both the master processor and remote processor are empty. This will be the prevailing situation in the majority of cases when a system response is required. The measurements for minor cycles 1, 2, and 3 were made in just such an operational environment. Averaging these measurements yields an SRT of 3669 us. Correcting this value for the instrumentation overhead produces the following result:

System Response Time = 3619 us.

It should be emphasized that this figure represents the minimum system response time that can be expected. As noted above, the addition of even one or two other asynchronous requests can greatly increase this time. During this study, processor loads often caused the system response time to exceed one minor cycle. This is an important point for designers since it may not be assumed that a non-local task will always be activated during the same minor cycle in which it is scheduled.

4.2.3.5 SRT Mathematical Model

In this section we construct an approximation model in order to predict system response time as a function of the basic bus busy time. By basic bus busy time we mean the bus loading time (over and above nominal) imposed by the activities of the various minor cycles of figure 3.1-5. For instance, in minor cycle 0, two 16-word COMPOOLs are transmitted and this causes a basic bus busy time of 640 usec (for minor cycle 0). The other basic bus busy times are 0. usec for minor cycles 1, 2, 3, and 4, 1760 usec for minor cycle 5 and 1355 usec (estimated from section 4.2.1.4) for minor cycle 6.

To form the linear predictor we plot the measured system response times (refer to the previous section) versus the nominal bus busy time referred to above. This is shown in figure 4.2-5. The least squares straight line (regression line) is then fitted to the data. This is also shown in figure 4.2-5. We then use the regression line equation as our predictor: SRT = 1.002 (Nominal Bus Busy Time) + 3808 usec, where SRT is the estimated system response time.

This analysis shows that there is almost a one-for-one slide in SRT as a result of the additional bus activity imposed by the various tasks of figure 3.1-5. The curve in figure 4.2-5 would be expected to level out as more and more bus activity is added. This is due to the fact that eventually some of that bus activity will be queued after the scheduled request. This was not the case for this test.

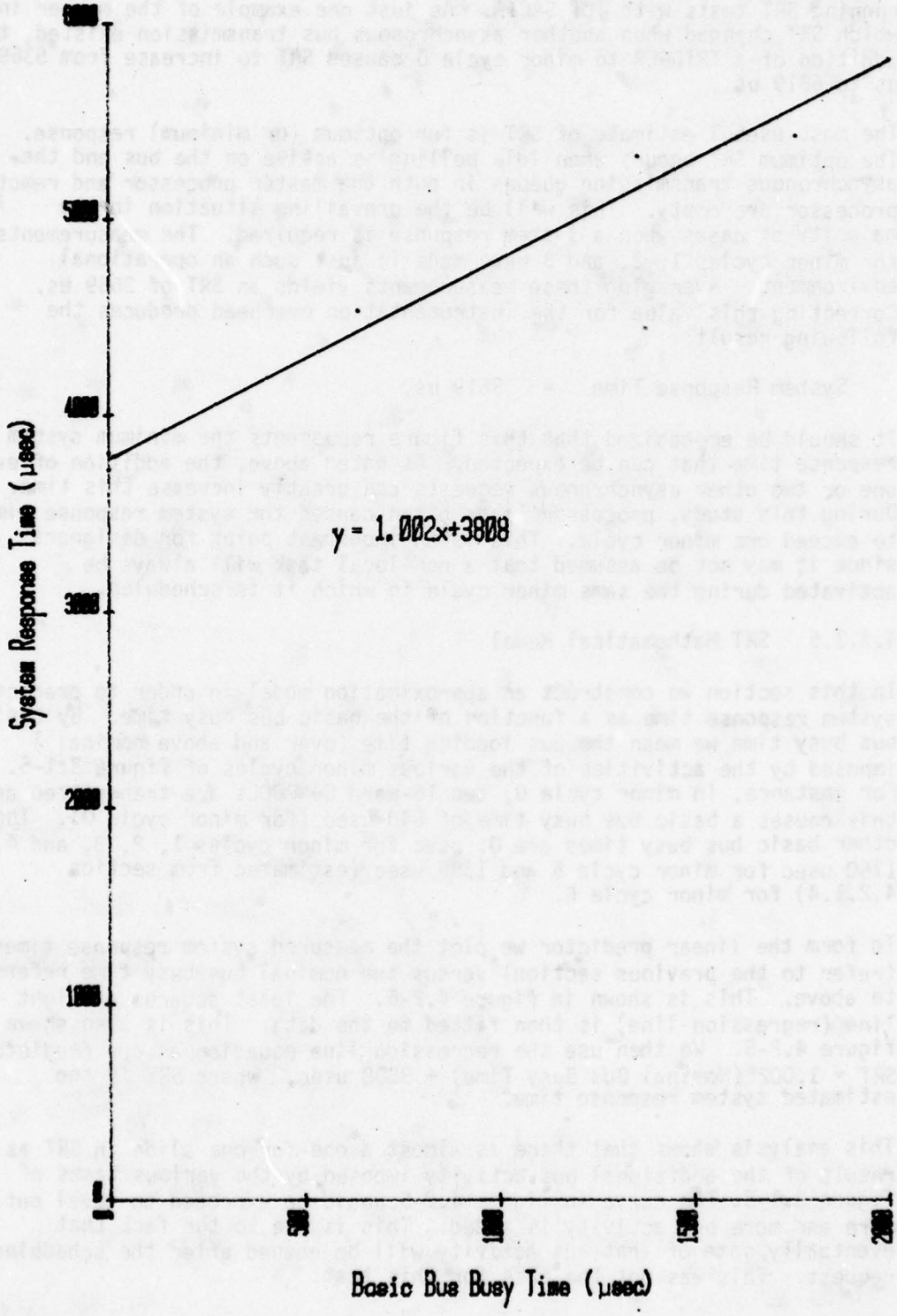


Figure 4.2-5 System Response Times

4.2.4 Test #4 - Event Service Overhead

Event Service Overhead (ESO) is defined as both the total amount of time spent in the executive handling event service and the amount of time spent handling the individual event services by type.

The events which were selected for the DAIS Executive Evaluation Event Service Overhead timing include:

- o Applications Task Signal Request
- o Privileged Task Signal Request
- o Applications Task Event Wait Request
- o Local Executive Task Event Handling
- o Local Executive Event Handling
- o Local Executive Minor Cycle Event Handling

For a more detailed description of the Event Service Overhead test, see the "Test Plan for the DAIS Executive Evaluation Program."

4.2.4.1 ESO Test Description

4.2.4.1.1 Instrumented Routines

As specified in the Test Plan, the ESO test is instrumented in the remote processor only. ESO processing is the same in the master processor, but the data is more difficult to extract because of the additional interrupts in the master processor.

Due to the design of the DAIS Executive Program, the gathering of ESO data cannot be accomplished by instrumenting the executive at the entry to and exit from an event service routine because during the handling of the event, an interrupt may occur causing control to pass to another executive routine rather than return to the requesting routine. When this occurs the time spent servicing the interrupt must be taken into account rather than treating it as event service overhead. Additionally, the return to the requesting task may not be effected due to the event service. In this case the time must be noted when the transfer of control takes place. To accomplish this the executive routine X$REM1 is instrumented at the three points where transfer of control is effected.

- o X$SUSP
- o X$CALL
- o X$REST

In addition to these instrumentation hooks, the routines which handle the event service types listed above will be instrumented. These routines are:

- o X$ASIG
- o X$PSIG
- o X$AWTE
- o X$TEVH
- o X$EVHA
- o X$MCSE

The actual instrumentation in each of these routines is described below.

X$SUSP Instrumentation

Routine X$SUSP is instrumented since event service is complete at this point.

Location	Contents	Description of Contents
*M X$SUSP + X'14'**	70F0	JC 15,
M X$SUSP + X'15'	5154	ESOR14

The instruction which is overstored at X$SUSP + X'14' is saved for execution prior to return to X$SUSP.

Location	Contents	Description of Contents
M ESOSV14	50F0	Overstored instruction
M ESOSV14+1	0495	at X$SUSP + X'14'

X$REST Instrumentation

Routine X$REST is instrumented at the point just prior to returning to the current task.

Location	Contents	Description of Contents
M X$REST + X'0C'	70F0	JC 15,
M X$REST + X'0D'	529A	LEOR20

The instruction which is overstored at X$REST + X'0C' is saved for execution prior to return to X$REST.

Location	Contents	Description of Contents
M LEOSV20	50F0	Overstored instruction
M LEOSV20+1	0495	at X$REST + X'0C'

X$CALL Instrumentation

Routine X$CALL is instrumented at the point just prior to jumping to the program to be executed.

Location	Contents	Description of Contents
M X$CALL + X'06'	70F0	JC 15,
M X$CALL + X'07'	513C	ESOR13

The instruction which is overstored at X$CALL + X'06' is saved for execution prior to return to X$CALL.

*"M" signifies a modification of the referenced location.
**"X" signifies an offset.

Location	Contents	Description of Contents
M ESOSV13	80FF	Overstored instruction
M ESOSV13+1	FFE8	from X$CALL + X'06'

X$ASIG Instrumentation

Routine X$ASIG is instrumented at only one point, the entry, because there is never a return to the requesting program from X$ASIG.

Location	Contents	Description of Contents
M X$ASIG + X'0E'	7220	JS 2,
M X$ASIG + X'0F'	508C	ESOR1

The instruction which is overstored at X$ASIG + X'0E' is saved for execution prior to return to routine X$ASIG.

Location	Contents	Description of Contents
M ESOSV1	801F	Overstored instruction
M ESOSV1+1	0001	from X$ASIG + X'0E'

X$PSIG Instrumentation

Routine X$PSIG is instrumented at both the entry and exit points since it is called from a privileged mode task and return after interrupt service will be to the privileged mode task.

Location	Contents	Description of Contents
M X$PSIG + X'2'	7220	JS 2,
M X$PSIG + X'3'	50AA	ESOR3
M X$PSIG + X'18'	70F0	JC 15,
M X$PSIG + X'19'	50BA	ESOR4

The instruction which is overstored at X$PSIG + X'2' is saved for execution prior to return to X$PSIG. The instruction at X$PSIG + X'18' is not saved since this instruction is a return via general register 2. This instruction is duplicated in routine ESOR4.

Location	Contents	Description of Contents
M ESOSV3	801F	Overstored instruction
M ESOSV3+1	0001	from X$PSIG + X'2'

X$AWTE Instrumentation

Routine X$AWTE is instrumented only at the entry point because there is never a return to the requesting program from X$AWTE.

Location	Contents	Description of Contents
M X$AWTE + X'0E'	7220	JS 2,
M X$AWTE + X'0F'	50C8	ESOR5

The instruction which is overstored at X$AWTE + X'0E' is saved for execution prior to return to X$AWTE.

Location	Contents	Description of Contents
M ESOSV5	801F	Overstored instruction
M EXOSV5+1	0001	from X$AWTE + X'0E'

X$TEVH Instrumentation

Routine X$TEVH is instrumented at the entry and exit points. Since it operates in the privileged mode it will run to completion.

Location	Contents	Description of Contents
M X$TEVH + X'2'	7220	JS 2,
M X$TEVH + X'3'	50E6	ESOR7
M X$TEVH + X'7C'	70F0	JC 15,
M X$TEVH + X'7D'	50F6	ESOR8

The instruction which is overstored at X$TEVH + X'2' is saved for execution prior to return to XSTEVH. The instruction at X$TEVH + X'7C' is a jump to the address in R2 (Return) which is effected in the EES code, therefore it is not saved.

Location	Contents	Description of Contents
M ESOSV7	801F	Overstored instruction
M ESOSV7+1	0000	from X$TEVH + X'2'

X$EVHA Instrumentation

Routine X$EVHA is instrumented at the entry and exit points. Since it operates in privileged mode it will run to completion.

Location	Contents	Description of Contents
M X$EVHA + X'2'	7220	JS 2,
M X$EVHA + X'3'	5104	ESOR9
M X$EVHA + X'EA'	70F0	JC 15,
M X$EVHA + X'EA'	5114	ESOR10

The instruction which is overstored at X$EVHA + X'2' is saved for execution prior to the return to X$EVHA. The instruction at X$EVHA + X'EA' is not saved since it is a return via register 2 which is duplicated in the EES code.

Location	Contents	Description of Contents
M ESOSV9	801F	Overstored instruction
M ESOSV9+1	0000	from X$EVHA + X'2'

X$MCSE Instrumentation

Routine X$MCSE is instrumented at the entry and exit points because it operates in privileged mode and will run to completion.

Location	Contents	Description of Contents
M X$MCSE + X'2'	7220	JS 2,
M X$MCSE + X'3'	5122	ESOR11
M X$MCSE + X'186'	70F0	JC 15,
M X$MCSE + X'187'	5130	ESOR12

The instruction which is overstored at X$MCSE + X'2' is saved for execution prior to the return to X$MCSE. The instruction X$MCSE + X'186' is not saved since it is a return via register 2 which is duplicated in the EES code.

Location	Contents	Description of Contents
M ESOSV11	801F	Overstored instruction
M ESOSV11+1	0000	from X$MCSE + X'2'

4.2.4.1.2 Factors Affecting Event Service Overhead - ESO

In order to assess the factors which affect the ESO the measurement must be defined. In this test the total ESO was measured as well as the individual event service types. Since the total is in fact just a summation of the individual event service times, only the individual event service measurements will be examined.

Sequence of Events:

Signal (Within Local Processor)
1. Applications program issues a signal.

2. Signal service routine calls the event service routine.

Signal (From Non-local Processor)
1. Asynchronous message is received and queued.
2. Local executive recognizes the interprocessor signal and calls the event handling routine.

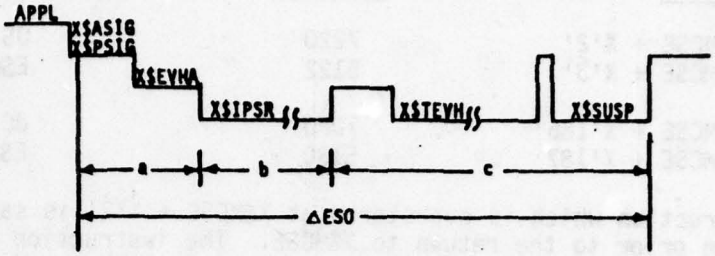
WAIT (Absolute Time)
1. Applications task calls the absolute time wait routine.
2. Task is suspended and entered into time wait queue.

WAIT (For Event)
1. Applications program calls event wait routine.
2. Task is suspended and entered into wait queue.

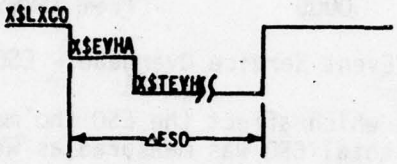
Minor Cycle
1. Master function mode command is received and interrupts the processor.
2. Minor cycle number is stored and minor cycle event flag is set.
3. Interrupt is dismissed.
4. Local executive recognizes minor cycle event and calls minor cycle setup routine.

4.2.4.1.3 ESO Time Line

Signal (Within Local Processor)

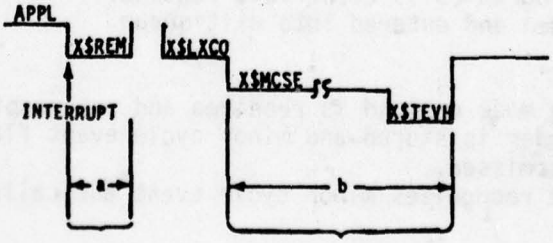


Signal (From Nonlocal Processor)



WAIT

Minor Cycle



a+b = ESO

4.2.4.1.4 Factors Affecting Lengths of the Time Lines

The items a - c below refer to segments of the above time line.

SIGNAL (Within Local Processor)

a. fixed

b. dependent upon the number of non-local copies of the event.

c. dependent upon the number of tasks waiting for the event and the number of tasks activated by the event that have task activation events.

SIGNAL (From Non-local Processor)

dependent upon the number of tasks waiting for the event and the number of tasks activated by the event which have task activation events

WAIT

dependent upon whether this is the only task event in the queue (the actual difference is negligible).

Minor Cycle

a. fixed and measured in ISO test #2.

b. dependent upon the number of tasks waiting for this minor cycle event and number of tasks activated by this minor cycle event which have task activation events.

4.2.4.2 ESO Instrumentation Timing

These estimates are based upon the instruction times provided in "DAIS Processor Instruction Set," SA 401301.

X$SUSP (at end of ESO Timing)

22.0 usec (if timing is not active)
57.8 usec (if timing is active)

X$REST (at end of ESO Timing)

22.0 usec (if timing is not active)
57.8 usec (if timing is active)

X$CALL (at end of ESO Timing)

27.2 usec (if timing is not active)
63.0 usec (if timing is active)

X$ASIG (at start of ESO Timing) - 51.0 usec

X$PSIG (at start of ESO Timing) - 51.0 usec

X$PSIG (at end of ESO Timing) - 48.6 usec

X$AWTE (at end of ESO Timing) - 51.0 usec

X$TEVH (at start of ESO Timing) - 51.0 usec

X$TEVH (at end of ESO Timing) - 48.6 usec

X$EVHA (at start of ESO Timing) - 51.0 usec

X$EVHA (at end of ESO Timing) - 48.6 usec

X$MCSE (at start of ESO Timing) - 51.0 usec

X$MCSE (at end of ESO Timing) - 48.6 usec

4.2.4.3 ESO Test Control Tables

Test Control Tables SACIA and PINS2 were used to collect data for the ESO tests. A description of these Test Control Tables can be found in paragraphs A.2.1 and A.2.2 of appendix A.

4.2.4.4 ESO Performance Data

During the ESO tests, timing data was collected for four local executive processes: the event handler, signal, event wait, and minor cycle setup. The event handler and the signal processes require varying amounts of time depending upon the destination of the event in question. The time required to process an event which must be sent over the bus is not the same as the time for an event which is local to the processor. A summary of the times measured for the event handler, signal, and event wait is given in table 4.2-5. As implemented in PINS, there is only one task waiting for each event and one task activated as the result of the event.

Executive Process	Measured Time
Event Handler - Generate Local Signal	252 us
Event Handler - Generate Signal for Another Processor	340 us
Event Handler - Receive Signal from Another Processor	233 us
Signal - For Local Processor	469 us
Signal - For Another Processor	563 us
Event Wait	353 us
(Refer to table 4.2-7 for times corrected for overhead)	

Table 4.2-5 Event Service Overhead Data

The data listed in table 4.2-5 are not independent. Since the signal routine always calls the event handler, the times measured for the signal routine include the times for the corresponding measurement on the event handler. In particular, the local signal time of 469 us includes the 252 us spent in the event handler, and the global signal time of 563 us includes the 340 us spent in the event handler. Since the time of interest to a system designer is the total time spent in servicing the signal request, the fractional times represented by the first two entries for the event handler will not be carried forward to table 4.2-7. The third entry for the event handler provides the time spent in servicing a signal received from another processor and this entry is carried forward to table 4.2-7.

In order to correct the signal processing times for instrumentation overhead, it must be remembered that the measured times include two sets of instrumentation hooks. The first set of instrumentation hooks records the entrance into the signal routine itself while the second set records both the entrance to and the exit from the event handler. The corrected times given in table 4.2-7 reflect the elimination of the extra instrumentation hooks (used to trap the event handler service time) from the signal processing time.

The final item studied in the ESO testing is the minor cycle setup time. Since the minor cycle setup time depends in part on the number of tasks being activated during that minor cycle, TCT SACIA was used to monitor several minor cycles which had varying numbers of tasks being activated in different test phases. Table 4.2-6 shows the data that were obtained. The first thing that is evident about this data is that the base load setup time reported in minor cycle 1 is inconsistent with the values reported for other minor cycles. This is because four privileged mode tasks are activated and run in minor cycle 1. Table 4.2-6 can be used to find the difference between the setup time in the base load and the setup time when extra tasks are added.

Minor Cycle	Setup Time In Base Load	Setup Time When Extra Tasks Are Added		
		1 Extra Task	2 Extra Tasks	3 Extra Tasks
0	280 us			
1	1127 us		1427 us	1506 us
2	522 us		675 us	742 us
3	360 us			
4	271 us			
5	330 us			
6	364 us	436 us		
7	269 us			

Table 4.2-6 Task Activation Time Data

These differences (delta setup times) can be plotted as a function of the number of additional tasks to determine the average length of time necessary to activate one additional task during minor cycle setup (figure 4.2-6). This can be determined by taking the differences in the reported values and averaging the numbers (i.e., determining the slope of the straight line in figure 4.2-6). The result obtained is:

Task Activation Time - 74 us

Data for minor cycle 1 is shown separately since it does not follow the pattern established by the other minor cycles.

With the exception of the task activation time, all of the ESO measurements summarized above include the instrumentation overhead. These measurements were corrected for the overhead incurred, and the corrected event service times are shown in table 4.2-7.

Executive Process	Corrected Time
Signal - Receive From Another Processor	183 us
Signal - To Another Processor	413 us
Signal - To Local Processor	319 us
Event Wait	303 us
Task Activation	74 us

Table 4.2-7 Event Service Overhead Times

The data in table 4.2-7 are shown graphically in figure 4.2-7.

4.2.5 Test #5 - Master Executive Overhead

Master Executive Overhead (MEO) is defined as the total amount of time used to perform the bus control and system control services in the master processor. This time excludes any local executive overhead in this same processor.

For a more detailed discussion of the master executive overhead test see the "Test Plan for the DAIS Executive Evaluation Program."

4.2.5.1 MEO Test Instrumentation

The majority of processing in the master executive computer program is accomplished in routine M$BCON in response to various system requests. Therefore, M$BCON is instrumented at the entry and exit points. Additional master executive processing occurs in the local executive interface routines M$ACB and M$ASI so these routines are also instrumented at the entry and exit points.

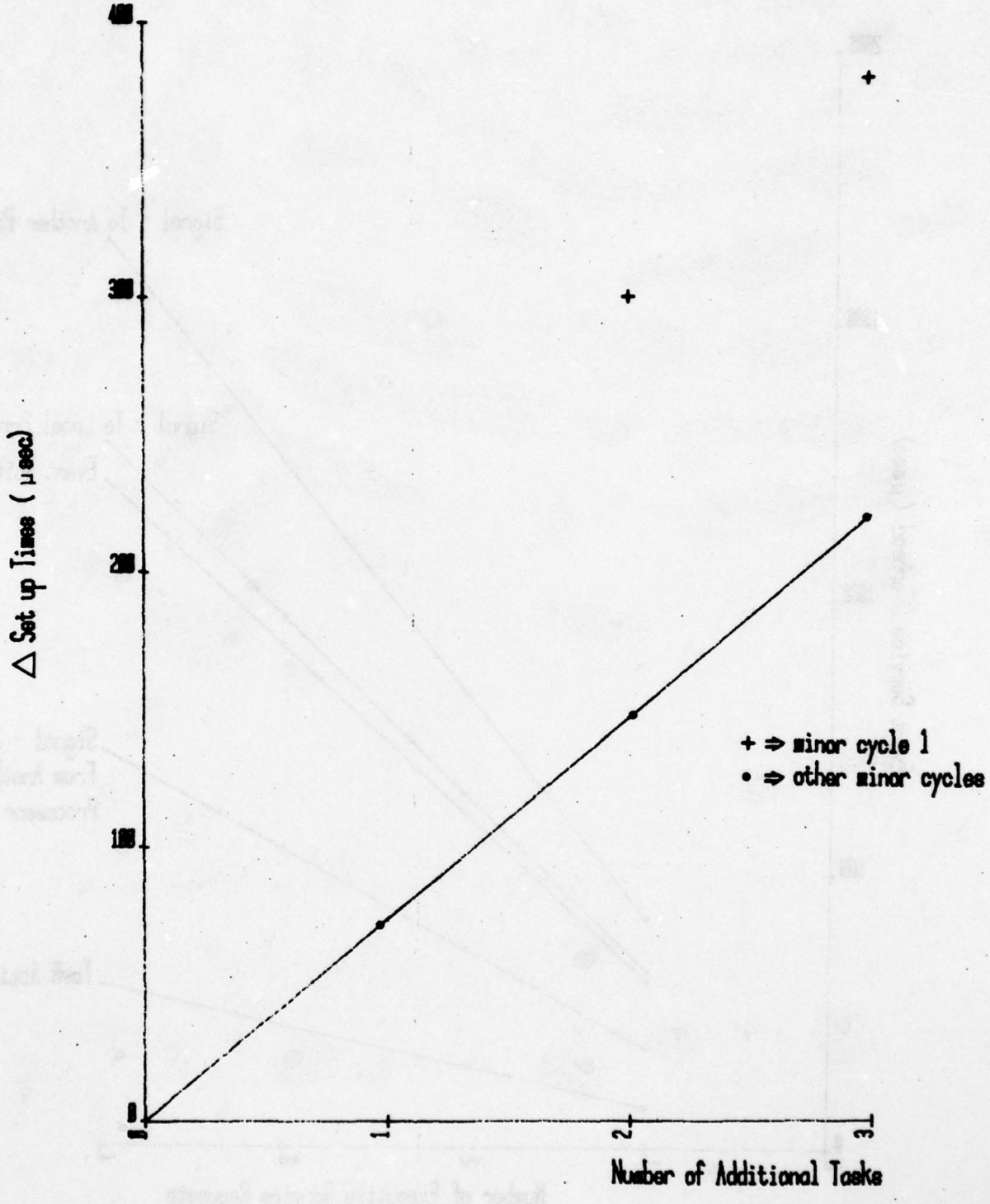


Figure 4.2-6 Task Activation Time Data

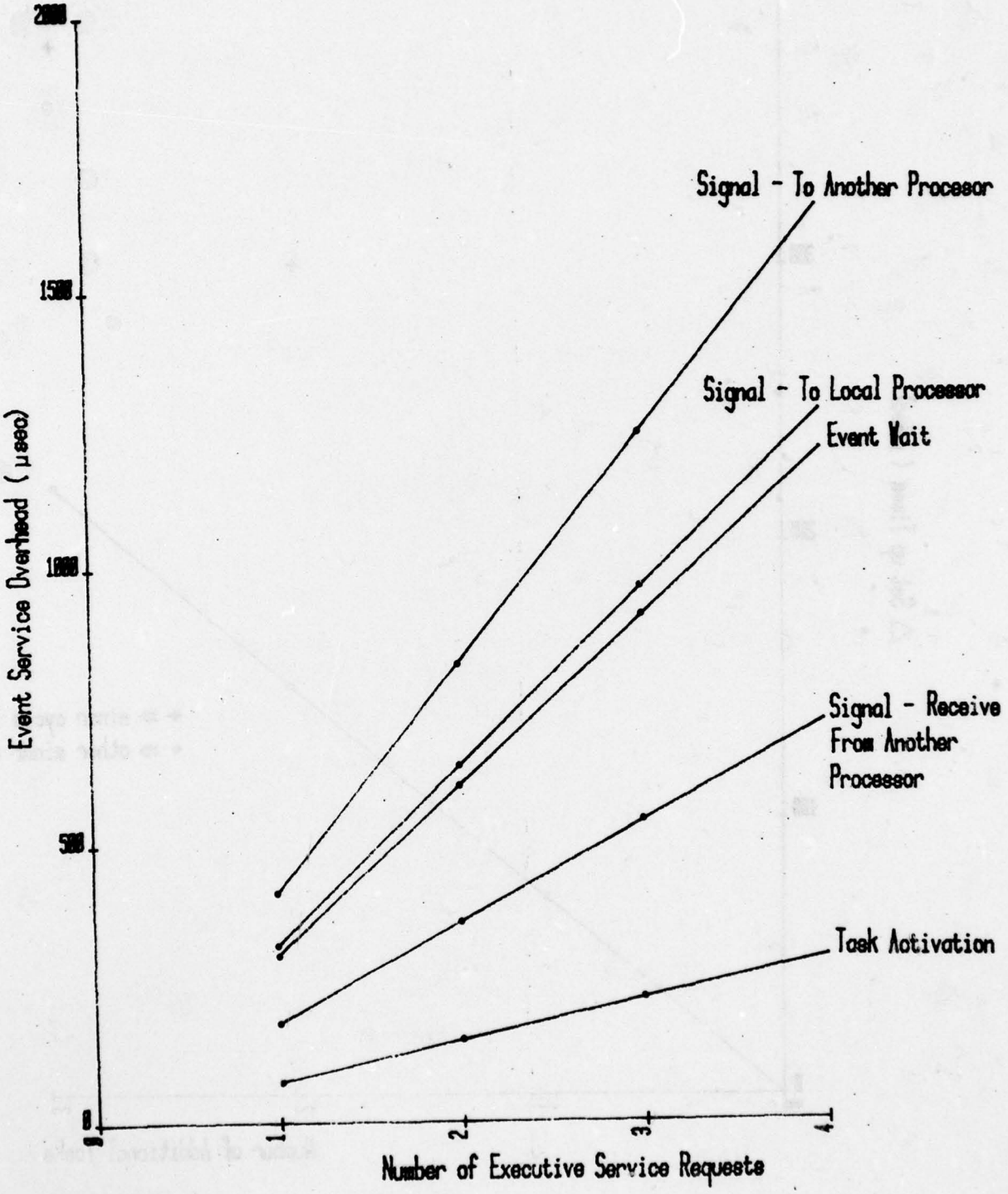


Figure 4.2-7 Event Service Overhead Times

Since almost all of the master executive overhead processing is in response to an interrupt of some sort, the data must be closely correlated with the interrupt service overhead for the master processor. Also since M$BCON operates with interrupts enabled most of the time, the correlated interrupt service overhead data must be extracted from the master executive overhead data.

M$ASI Instrumentation

Routine M$ASI is instrumented at the entry and exit points since it is called by a privileged mode local executive routine and the return will be to that routine.

Location	Contents	Description of Contents
*M M$ASI + X'2'**	7220	JS 2,
M M$ASI + X'3'	7F3A	MEOM1
M M$ASI + X'14'	70F0	JC 15,
M M$ASI + X'15'	7F48	MEOM2

The instruction which is overstored at M$ASI + X'2' is saved for execution prior to the return to M$ASI. The instruction at M$ASI + X'14' is not saved since it is a return via register 2 and is duplicated in the EES code.

Location	Contents	Description of Contents
M MEOSV1	801F	Overstored instruction
M MEOSV1+1	0000	from M$ASI + X'2'

M$ACB Instrumentation

Routine M$ACB is instrumented at the entry and exit points since it is called by a privileged mode local executive routine and return will be to that routine.

Location	Contents	Description of Contents
M M$ACB + X'02'	7220	JS 2,
M M$ACB + X'03'	7F56	MEOM3
M M$ACB + X'8E'	70F0	JC 15,
M M$ACB + X'8F'	7F64	MEOM4

The instruction which is overstored at M$ACB + X'2' is saved for execution prior to the return to M$ACB. The instruction at M$ACB + X'8E' is not saved since it is a return via register 2 and is duplicated in the EES code.

*"M" signifies a modification of the referenced location.
**"X" signifies an offset.

Location	Contents	Description of Contents
M MEOSV3	801F	Overstored instruction
M MEOSV3+1	0000	from M$ACB + X'2'

M$BCON Instrumentation

Routine M$BCON is instrumented at the entry and exit points since, even though it is interruptable, it will always run to completion.

Location	Contents	Description of Contents
M M$BCON + X'50E'	7220	JS 2,
M M$BCON + X'50F'	7F72	MEOM5
M M$BCON + X'5DA'	70F0	JC 15,
M M$BCON + X'5DB'	7F8E	MEOM7

The instruction which is overstored at M$BCON + X'50E' is saved for execution prior to the return to M$BCON. The instruction at M$BCON + X'5DA' is not saved since it is a return via register 2 and is duplicated in the EES code.

Location	Contents	Description of Contents
M MEOSV5	9000	Overstored instruction
M MEOSV5+1	0C7C	from M$BCON + X'50E'

4.2.5.2 MEO Instrumentation Timing

These estimates are based upon the instruction times provided in "DAIS Processor Instruction Set," SA 401301.

M$ASI (at start of Timing) - 48.8 usec

M$ASI (at end of Timing) - 45.8 usec

M$ACB (at start of Timing) - 48.8 usec

M$ACB (at end of Timing) - 45.8 usec

M$BCON (at start of Timing) - 49.2 usec

MSBCON (at end of Timing) - 46.2 usec

4.2.5.3 MEO Test Control Table

Test Control Tables PINS3 and PINS4 were used to generate data for the MEO tests. A description of these test control tables can be found in paragraphs A.2.3 and A.2.4 of appendix A.

4.2.5.4 MEO Performance Data

During the master executive overhead tests, timing data was collected for four master executive routines: fielding a timer A interrupt (M$TIMA), adding a critically timed block to the queue (M$ACB), the local executive asynchronous interface (M$ASI), and the bus control routine (M$BCON). Since M$TIMA is called by the interrupt handler whenever an interrupt 10 is serviced, the times measured for the M$TIMA routine were essentially the same as those measured for the interrupt 10. Since the interrupt 10 times are already listed in paragraph 4.2.2.4, the M$TIMA data will not be repeated here. The M$ASI measurements were made for SIGNALs being transmitted to the remote processor. The times measured for M$ASI did not appear to be heavily dependent on bus activity; the same results were obtained no matter when the executive request occurred.

The length of time required in M$BCON is dependent upon the interrupt which immediately preceded it. M$BCON time is reported separately for the four interrupts (1, 3, 5, and 10) which it follows. The M$BCON interval that follows an interrupt 3 can vary in length. When the interrupt 3 was generated following a TRIGGER completion, the M$BCON processing time was less than normal.

In a similar manner, the M$BCON interval that follows an interrupt 10 varies in length. Processing for an interrupt 10 which marks the precise time for a TRIGGER transmission (critically timed message) takes slightly longer than processing for an interrupt 10 which initiates a new minor cycle. The performance data showed this difference in processing time to be less than 10 us and the data are combined in a single interrupt 10 entry in table 4.2-8.

The data collected in the MEO measurements are summarized in the table below. These measurements include the instrumentation overhead which was removed to obtain the corrected times shown in table 4.2-8.

Executive Process	Measured Time	Corrected Time
M$BCON - Interrupt 1	353 us	310 us
M$BCON - Interrupt 3, normal	388 us	345 us
M$BCON - Interrupt 3, following TRIGGER	335 us	292 us
M$BCON - Interrupt 5	507 us	464 us
M$BCON - Interrupt 10	383 us	340 us
M$ASI	1081 us	1038 us
M$ACB	176 us	133 us

Table 4.2-8 Master Executive Overhead Times

It is interesting to note that the increased processing time required to service an interrupt 10 when a trigger is queued is almost entirely due to the addition of an M$ACB routine to the process. As noted in paragraph 4.2.2.4, the normal processing time for an interrupt 10 of 183 us jumps to 319 us when a TRIGGER is queued. The difference of 136 us compares very well with the M$ACB time of 133 us reported above.

By combining the processing times reported in table 4.2-8 with the master executive interrupt times presented in paragraph 4.2.2.4.1, the total time required for the master processor to service an applications "event" may be estimated. These events follow two different processing paths in the executive, one path for TRIGGERs and another for all other events (e.g., SCHEDULE, CANCEL, WRITE, SIGNAL). In addition, requests received from the remote processor are handled somewhat differently from requests received from the local executive in the master processor. For all requests (except TRIGGERs) received from a remote processor, the master executive fields one interrupt 5 and services it through M$BCON, and then, when the transmission is completed, fields one interrupt 3 and services it through M$BCON. The total master executive processing time for these events is:

Interrupt 5	355 us
M$BCON - Interrupt 5	464 us
Interrupt 3	238 us
M$BCON - Interrupt 3	345 us
Total Processing Time	1402 us

For requests received from the local executive in the master processor, the local executive interfaces with the master executive by calling M$ASI. When the transmission is completed, the master executive must then field the interrupt 3 and service it through M$BCON, as above. The total master executive processing time for these events is:

M$ASI	1038 us
Interrupt 3	238 us
M$BCON - Interrupt 3	345 us
Total Processing Time	1621 us

Servicing a TRIGGER is more complicated since TRIGGER processing is spread over several minor cycles. Data collected for executive activity in the first minor cycle (when the TRIGGER is queued) and the last minor cycle (when the TRIGGER is sent) will apply to any TRIGGER generated in the system. The total amount of executive time required to service a TRIGGER will vary however, since for each minor cycle that the TRIGGER is queued, the M$ACB routine must be activated. In PINS, a TRIGGER COMPOOL block is transmitted over the bus three minor cycles after the TRIGGER request is received by the local executive. For a TRIGGER generated in the remote processor by PINS, the timing in the master executive breaks down as shown in table 4.2-9.

Minor Cycle	Executive Routine	Time	
1	Interrupt 5	355 us	
1	M$BCON - Interrupt 5	464 us	
1	M$ACB	133 us	1668 us
1	Interrupt 3	238 us	
1	M$BCON - Interrupt 3	345 us	
1	M$ACB	133 us	
2	M$ACB	133 us	133 us
3	M$ACB	133 us	133 us
4	Interrupt 10	147 us	
4	M$BCON - Interrupt 10	340 us	1017 us
4	Interrupt 3	238 us	
4	M$BCON - Interrupt 3	292 us	
Total TRIGGER Time			2951 us

Table 4.2-9 TRIGGER Overhead Time

This TRIGGER time of 2951 us has meaning only for the particular case when the TRIGGER is scheduled to be sent three minor cycles after it is queued. The TRIGGER timing estimate can be made applicable to the general case by subtracting out the M$ACB times in minor cycles 2 and 3 and treating them separately. The result is:

TRIGGER Time = 2685 us + (n-1)133 us,

where n is the number of minor cycles specified in the TRIGGER statement.

The processing time for a TRIGGER originating in the master processor is the same except the initial interrupt 5 and its M$BCON processing are replaced with a M$ASI routine. For a TRIGGER generated by the local executive in the master processor, the processing time in the master executive is:

TRIGGER Time = 2904 us + (n-1)133 us,

where n is the number of minor cycles specified in the TRIGGER statement.

The processing times reported above cover all cases when the master executive must service a request from a local executive. In summary, the master executive service times are shown in table 4.2-10. A graphical representation is shown in figures 4.2-8 and 4.2-9.

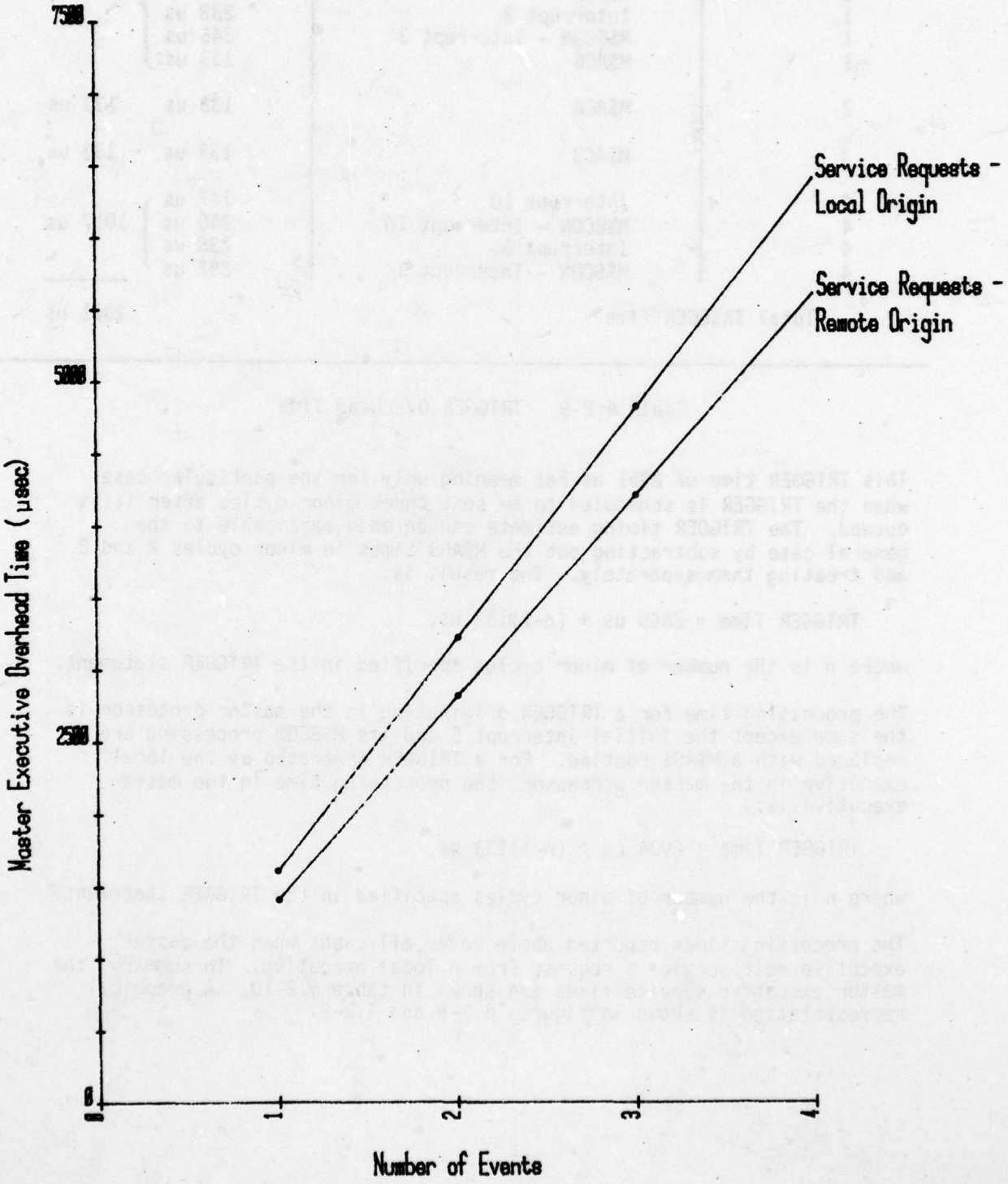


Figure 4.2-8 Master Executive Overhead Time for Non-Trigger Events

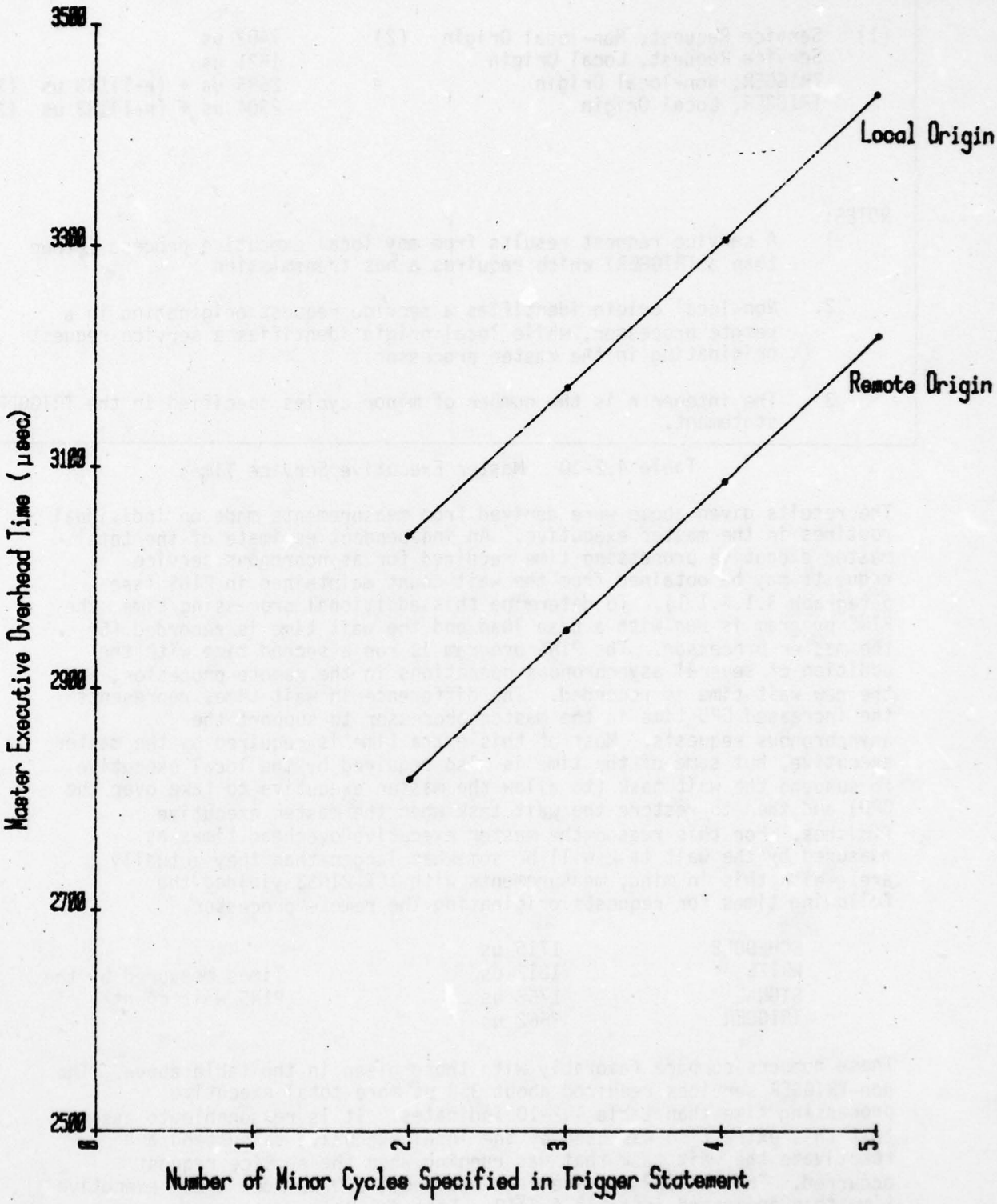
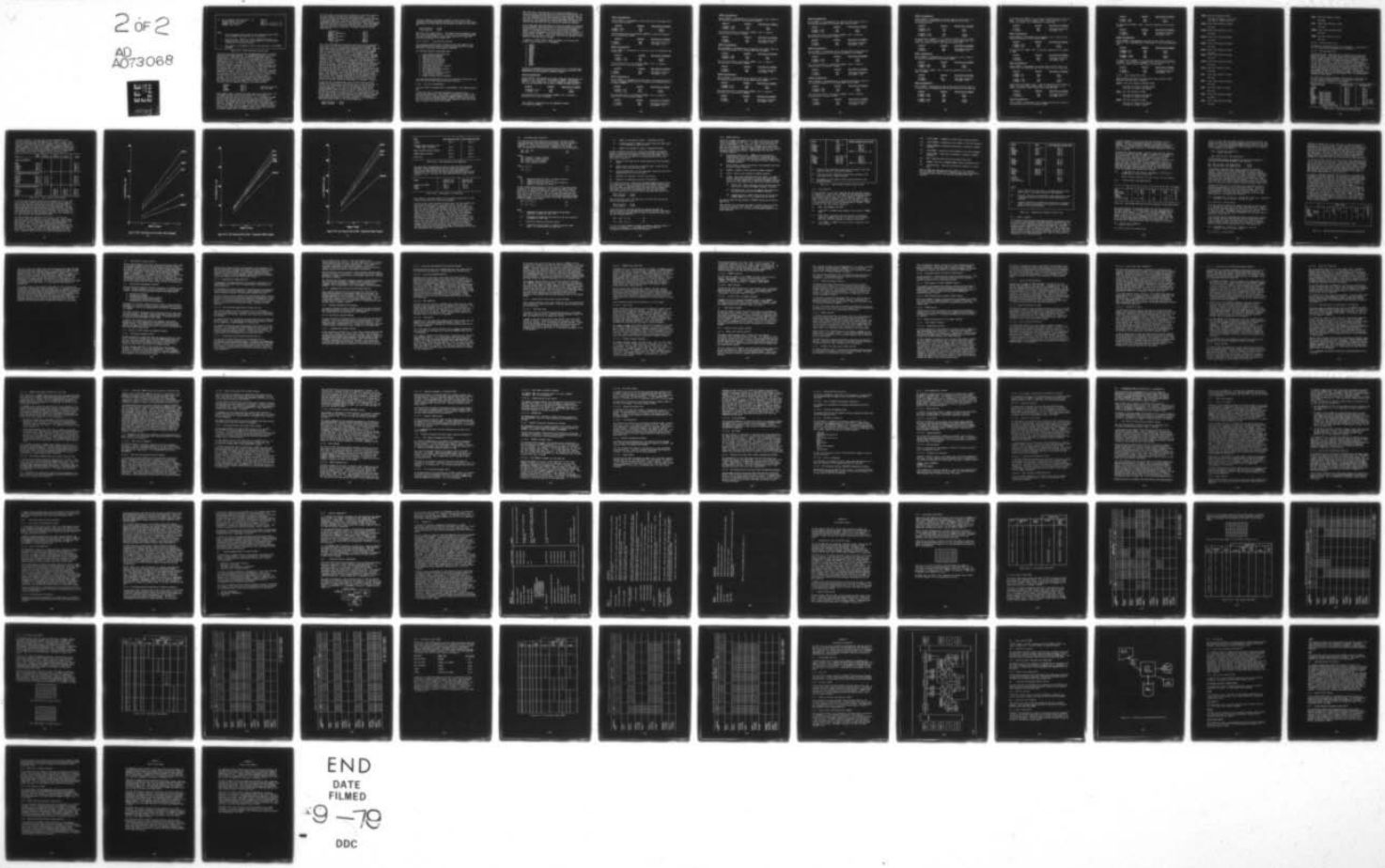


Figure 4.2-9 Master Executive Overhead Times For Triggers

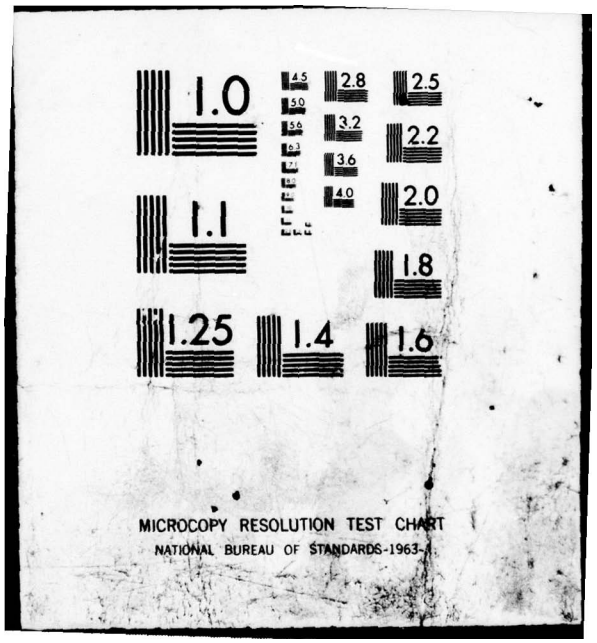
AD-A073 068 BOEING AEROSPACE CO SEATTLE WA BOEING MILITARY AIRPL--ETC F/G 17/7
EVALUATION OF DAIS TECHNOLOGY APPLIED TO THE INTEGRATED NAVIGAT--ETC(U)
MAY 79 D DEWEY, R BOUSLEY, S BEHNEN, J MASON F33615-77-C-1233

UNCLASSIFIED AFAL-TR-79-1061 NL

2 OF 2
AD A073068



END
DATE FILMED
9-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(1)	Service Request, Non-local Origin	(2)	1402 us
	Service Request, Local Origin		1621 us
	TRIGGER, Non-local Origin		2685 us + (n-1)133 us (3)
	TRIGGER, Local Origin		2904 us + (n-1)133 us (3)

NOTES:

1 A service request results from any local executive process (other than a TRIGGER) which requires a bus transmission

2. Non-local origin identifies a service request originating in a remote processor, while local origin identifies a service request originating in the master processor.

3 The integer n is the number of minor cycles specified in the TRIGGER statement.

Table 4.2-10 Master Executive Service Times

The results given above were derived from measurements made on individual routines in the master executive. An independent estimate of the total master executive processing time required for asynchronous service requests may be obtained from the wait count maintained in PINS (see paragraph 3.1.4.1.1). To determine this additional processing time, the PINS program is run with a base load and the wait time is recorded for the master processor. The PINS program is run a second time with the addition of several asynchronous operations in the remote processor, and the new wait time is recorded. The difference in wait times represents the increased CPU time in the master processor to support the asynchronous requests. Most of this extra time is required by the master executive, but some of the time is also required by the local executive to suspend the wait task (to allow the master executive to take over the CPU) and then to restore the wait task when the master executive finishes. For this reason the master executive overhead times as measured by the wait task will be somewhat larger than they actually are. With this in mind, measurements with TCT PINS3 yielded the following times for requests originating the remote processor.

SCHEDULE	1715 us	
WRITE	1812 us	Times measured by the
SIGNAL	1758 us	PINS wait count
TRIGGER	3562 us	

These numbers compare favorably with those given in the table above. The non-TRIGGER services required about 350 us more total executive processing time than table 4.2-10 indicates. It is reasonable to assume that this extra time was used by the local executive to suspend and reactivate the wait task that was running when the service request occurred. The TRIGGER service required about 600 us more total executive time than indicated in table 4.2-10. This is also a reasonable number since, for a TRIGGER, the local executive must twice suspend and restore the wait task - once when the TRIGGER is queued, and a second time when the TRIGGER COMPOOL block is transmitted over the bus.

Up to this point, the discussion of processing in the master executive has been primarily concerned with asynchronous events. It is also possible to examine in some detail the synchronous processing performed in the master executive. The master executive is responsible for synchronizing the minor cycles in every processor and for setting up the synchronous instruction list for a new minor cycle. These requirements lead to a fixed set of processing which must occur during every minor cycle that the system is operating. The master executive routines exercised are:

Interrupt 10	183 us
M$BCON - Interrupt 10	340 us
Interrupt 1	176 us
M$BCON - Interrupt 1	310 us
Interrupt 1	176 us
M$BCON - Interrupt 1	310 us
Total Time	1495 us

This is the time required by the master executive to set up each minor cycle, but it is still not the total master executive overhead that exists when there are no asynchronous requests to be serviced. There are other factors which contribute to the total MEO for an "idle" system (that is, a system which is not generating any asynchronous requests. The idle load in the master executive is the amount of processing required for a system even if it has no applications tasks). One major source of processing time which has not been measured is the interface between the master and local executives. It is estimated that this interface could be responsible for another 200 us of processing time in each minor cycle. Another source of overhead is the idle polling sequence on the bus. To determine how much overhead can be traced to the idle polling process, two sets of measurements were made on the system when all of the applications tasks were removed. The first set of measurements was made with the bus operating normally and the second set was taken after the normal idle polling sequence was changed to a single pass polling sequence. Processing time decreased by 293 us in each minor cycle. When the master-local interface and idle polling factors are added to the setup time, the revised MEO for an idle system is 1992 us. This figure is still somewhat lower than the total MEO which will be incurred under actual operating conditions since not every executive instruction has been isolated for measurement in the Phase I studies.

As was done with the asynchronous tasks, the total master executive overhead can be measured by using the wait task in the PINS program. To find the idle load overhead, both the asynchronous tasks (which are controlled by inputs in the TCT) and the synchronous tasks (which transfer COMPOOL blocks in support of the bus transmissions in the synchronous instruction list) were disabled. This left the wait task as the only applications task operating during the measurement period. Thus the wait task was able to directly measure the local executive overhead in the remote processor and the sum of local executive overhead and master executive overhead in the master processor. The percentage of time each processor spent in idle load overhead was:

Master Processor - 37.5%
Remote Processor - 10.3%

Since the overhead in the master processor is just the sum of the overhead in the master executive and the local executive, we derive the percentage of the time spent by each executive to support an idle load to be:

Master Executive - 27.2%
Local Executive - 10.3%

Each minor cycle lasts 7812.5 us. The master executive overhead in each minor cycle is therefore 2125 us. This result is in excellent agreement with the estimate of 1992 us derived above, particularly since that estimate was expected to be slightly low.

4.2.6 Test #6 - Local Executive Overhead

Local Executive Overhead (LEO) is defined as the total amount of time spent performing local executive service functions and the time spent handling the individual local executive functions.

The functions chosen for measurement were:

- o Local Executive Control Function
- o Schedule Service Function
- o Cancel Service Function
- o Terminate Service Function
- o Event Wait Service Function
- o Time Wait Service Function
- o Signal Service Function
- o Privileged Signal Service Function
- o Read Service Function
- o Privileged Read Service Function
- o Write Service Function
- o Privileged Write Service Function
- o Trigger Service Function
- o Compool Broadcast Service Function

For a more detailed description of the local executive overhead test, see "Test Plan for the DAIS Executive Evalution Function."

4.2.6.1 LEO Test Instrumentation

The local executive overhead test is instrumented in the remote processor only.

In most cases the local executive overhead is a function of an applications program call. The service is performed in response to the call and a return is made to the calling program; however, this is not always the case since the DAIS executive design allows interrupts during most local executive services and the local executive service itself may cause a change in the state of the system.

When servicing an interrupt which occurs during performance of an executive service, the interrupt service time must be determined so that it can be extracted from the executive service time since this time is not really chargeable to the local executive service overhead. For this test, this will be accomplished by running the remote interrupt service overhead test using the same test control tables which are used to run this test and the results will be collated for analytical purposes.

By the same token, when the handling of a service request or an interrupt causes a change in the system state, the associated overhead should not be charged to the service request but to local executive overhead in general. To accomplish this, the local executive control (X$LXCO) routine will be instrumented at the entry point and the two possible exit points, X$REST and X$CALL. These three subroutines are not actually in X$LXCO but they are the only exit points from the local executive control function to the applciations software.

In addition to X$LXCO, X$SUSP, X$REST and X$CALL the following local executive service routines are instrumented:

- o X$ASCH
- o X$ACAN
- o X$ATRM
- o X$AWTE
- o X$AWTA
- o X$ASIG
- o X$PSIG
- o X$ARD
- o X$PRD
- o X$AWR
- o X$PWR
- o X$ATR
- o X$CBBR

The actual instrumentation for each of these routines is described below and the local executive overhead software routines are described in the documentation for the executive evaluation software.

X$LXCO Instrumentation

Routine X$LXCO is instrumented at the entry point only since the exit for purposes of this test will be either X$CALL or X$REST. Additionally, X$LXCO is a never ending loop, so the EES software has been designed so that only one measurement will be taken in X$LXCO for each true entry.

Location	Contents	Description of Contents
*M X$LXCO + X'2'**	7220	JS 2,
M X$LXCO + X'3'	42AE	LEOR21

The instruction which is overstored at X$LXCO + X'2' is saved for execution prior to return to X$LXCO.

*"M" signifies a modification of the referenced location.
**"X"signifies an offset.

X$CALL Instrumentation

Routine X$CALL is instrumented at a point just prior to the jump to the routine to be activated.

Location	Contents	Description of Contents
M X$CALL + X'6'	70F0	JC 15,
M X$CALL + X'7'	5238	LEOR14

The instruction which is overstored at X$CALL+6 is saved for execution prior to returning to X$CALL.

Location	Contents	Description of Contents
M LEOSV14	80FF	Overstored instruction
M LEOSV14+1	FFE8	from X$CALL + X'6'

X$REST Instrumentation

Routine X$REST is instrumented at a point just prior to returning to the current task.

Location	Contents	Description of Contents
M X$REST + X'0C'	70F0	JC 15,
M X$REST + X'0D'	529A	LEOR20

The instruction which is overstored at X$REST + X'0C' is saved for execution prior to returning to X$REST.

Location	Contents	Description of Contents
M LEOSV20	50F0	Overstored instruction
M LEOSV20+1	0495	from X$REST + X'0C'

X$SUSP Instrumentation

Routine X$SUSP is instrumented since upon entry, the requested executive service is complete whether control is returned to the requesting task or is passed to X$LXCO.

Location	Contents	Description of Contents
M X$SUSP + X'14'	70F0	JC 15,
M X$SUSP + X'15'	5250	LEOR15

The instruction which is overstored at X$SUSP + X'14' is saved for execution prior to returning to X$SUSP.

Location	Contents	Description of Contents
M LEOSV15	50F0	Overstored instruction
M LEOSV15+1	0495	from X$SUSP + X'14'

X$ASCH Instrumentation

Routine X$ASCH is instrumented only at the entry point since a return to the requesting program will never be made by X$ASCH.

Location	Contents	Description of Contents
M X$ASCH + X'0E'	7220	JS 2,
M XRASCH + X'0F'	5168	LEOR1

The instruction which is overstored at X$ASCH + X'0E' is saved for execution prior to return to X$ASCH.

Location	Contents	Description of Contents
M LEOSV1	801F	Overstored instruction
M LEOSV1+1	0001	from X$ASCH + X'0E'

X$ACAN Instrumentation

Routine X$ACAN is instrumented only at the entry point since a return to the requesting program will be through X$SUSP and not X$ACAN.

Location	Contents	Description of Contents
M X$ACAN + X'0E'	7220	JS 2,
M X$ACAN + X'0F'	5178	LEOR2

The instruction which is overstored at X$ACAN + X'0E' is saved for execution prior to returning to X$ACAN.

Location	Contents	Description of Contents
M LEOSV2	801F	Overstored instruction
M LEOSV2+1	0001	from X$ACAN + X'0E'

X$ATRM Instrumentation

Routine X$ATRM is instrumented only at the entry point since a return to the requesting program will be through X$SUSP and not X$ATRM.

Location	Contents	Description of Contents
M X$ATRM + X'0E'	7220	JS 2,
M X$ATRM + X'0F'	5188	LEOR3

The instruction which is overstored at X$ATRM + X'0E' is saved for execution prior to returning to X$ATRM.

Location	Contents	Description of Contents
M LEOSV3	801F	Overstored instruction
M LEOSV3+1	0001	from X$ATRM + X'0E'

X$AWTE Instrumentation

Routine X$AWTE is instrumented only at the entry point since a return to the requesting program will be through X$SUSP and not X$AWTE.

Location	Contents	Description of Contents
M X$AWTE + X'0E'	7220	JS 2,
M X$AWTE + X'0F'	5198	LEOR4

The instruction which is overstored at X$AWTE + X'0E' is saved for execution prior to returning to X$AWTE.

Location	Contents	Description of Contents
M LEOSV4	801F	Overstored instruction
M LEOSV4+1	0000	from X$AWTE + X'0E'

X$AWTA Instrumentation

Routine X$AWTA is instrumented only at the entry point since a return to the requesting program will be made through X$SUSP and not X$AWTA.

Location	Contents	Description of Contents
M X$AWTA + X'0E'	7220	JS 2,
M X$AWTA + X'0F'	51A8	LEOR5

The instruction which is overstored at X$AWTA + X'0E' is saved for execution prior to returning to X$AWTA.

Location	Contents	Description of Contents
M LEOSV5	801F	Overstored instruction
M LEOSV5+1	0001	from X$AWTA + X'0E'

X$ASIG Instrumentation

Routine X$ASIG is instrumented only at the entry point since a return to the requesting program will be through X$SUSP and not X$ASIG.

Location	Contents	Description of Contents
M X$ASIG + X'0E'	7220	JS 2,
M X$ASIG + X'0F'	51B8	LEOR6

The instruction which is overstored at X$ASIG + X'0E' is saved for execution prior to return of X$ASIG.

Location	Contents	Description of Contents
M LEOSV6	801F	Overstored instruction
M LEOSV6+1	0001	from X$ASIG + X'0E'

X$PSIG Instrumentation

Routine X$PSIG is instrumented at the entry and exit points since it is called from a privileged mode task and the return will be to the privileged mode task.

Location	Contents	Description of Contents
M X$PSIG + X'2'	7220	JS 2,
M X$PSIG + X'3'	51C8	LEOR7
M X$PSIG + X'18'	70F0	JC 15,
M X$PSIG + X'19'	527E	LEOR18

The instruction at X$PSIG + X'2' is saved for execution prior to return to X$PSIG. The instruction at X$PSIG + X'18' is not saved since it is a return via register 2 which is duplicated in the EES code.

Location	Contents	Description of Contents
M LEOSV7	801F	Overstored instruction
M LEOSV7+1	0001	from X$PSIG + X'2'

X$ARD Instrumentation

Routine X$ARD is instrumented only at the entry point since a return to the requesting program will not be made by X$ARD.

Location	Contents	Description of Contents
M X$ARD + X'0E'	7220	JS 2,
M X$ARD + X'0F'	51D8	LEOR8

The instruction at X$ARD + X'0E' is saved for execution prior to the return to X$ARD.

Location	Contents	Description of Contents
M LEOSV8	801F	Overstored instruction
M LEOSV8+1	0001	from X$ARD + X'0E'

X$PRD Instrumentation

Routine X$PRD is instrumented at both the entry and exit points since it is called from a privileged mode task and the return will be to that task.

Location	Contents	Description of Contents
M X$PRD + X'02'	7220	JS 2,
M X$PRD + X'03'	51E8	LEOR9
M X$PRD + X'56'	70F0	JC 15,
M X$PRD + X'57'	5264	LEOR16

The instruction at X$PRD + X'2' is saved for execution prior to return to X$PRD. The instruction at M X$PRD + X'56' is not saved since it is a return via register 2 and is duplicated in the EES code.

Location	Contents	Description of Contents
M LEOSV9	801F	Overstored instruction
M LEOSV9+1	0000	from X$PRD + X'2'

X$AWR Instrumentation

Routine X$AWR is instrumented at the entry point since a return to the requesting program will not be made by X$AWR.

Location	Contents	Description of Contents
M X$AWR + X'0E'	7220	JS 2,
M X$AWR + X'0F'	51F8	LEOR10

The instruction at X$AWR + X'0E' is saved for execution prior to return to X$AWR.

Location	Contents	Description of Contents
M LEOSV10	801F	Overstored instruction
M LEOSV10+1	0000	from X$AWR + X'0E'

X$PWR Instrumentation

Routine X$PWR is instrumented at the entry and exit points since it is called from a privileged mode task and the return will be to that task.

Location	Contents	Description of Contents
M X$PWR + X'2'	7220	JS 2,
M X$PWR + X'3'	5208	LEOR11
M X$PWR + X'9C'	70F0	JS 15,
M X$PWR + X'9D'	5272	LEOR17

The instruction at X$PWR + X'2' is saved for execution prior to return to X$PWR. The instruction at X$PWR + X'9C' is not saved since it is a return via register 2 which is duplicated in the EES code.

Location	Contents	Description of Contents
M LEOSV11	801F	Overstored instruction
M LEOSV11+1	0000	from X$PWR + X'2'

X$ATR Instrumentatioan

Routine X$ATR is instrumented only at the entry point since a return to the requesting task will not be made by X$ATR.

Location	Contents	Description of Contents
M X$ATR + X'0E'	7220	JS 2,
M X$ATR + X'0F'	5218	LEOR12

The instruction at X$ATR + X'0E' is saved for execution prior to return to X$ATR.

Location	Contents	Description of Contents
M LEOSV12	801F	Overstored instruction
M LEOSV12+1	0001	from X$ATR + X'0E'

X$CBBR Instrumentation

Routine X$CBBR is instrumented at the entry and exit points since it is called by a privileged mode task and the return will be to the requesting task.

Location	Contents	Description of Contents
M X$CBBR + X'2'	7220	JS 2,
M X$CBBR + X'3'	5228	LEOR13
M X$CBBR + X'136'	70F0	JC 15,
M X$CBBR + X'137'	528C	LEOR19

The instruction at X$CBBR + X'2' is saved for execution prior to return to X$CBBR. The instruction at X$CBBR + X'136' is not saved since it is a return via register 2 and is duplicated in the EES code.

Location	Contents	Description of Contents
M LEOSV13	801F	Overstored instruction
M LEOSV13+1	0000	from X$CBBR + X'2'

4.2.6.2 LEO Instrumentation Timing

These estimates are based on the instruction times provided in "DAIS Processor Instruction Set," SA 401301.

X$LXCO (start local executive overhead timing)

19.8 usec (if timing is already active)
55.8 usec (if timing is not active)

X$CALL (end local executive timing)

27.2 usec (if timing is not active)
63.0 usec (if timing is active)

X$REST (end local executive timing)

22.0 usec (if timing is not active)
57.8 usec (if timing is active)

X$SUSP (end local executive timing)

22.0 usec (if timing is not active)
57.8 usec (if timing is active)

X$ASCH (start local executive timing)

51.0 usec

X$ACAN (start local executive timing)

51.0 usec

X$ATRM (start local executive timing)

51.0 usec

X$AWTE (start local executive timing)

51.0 usec

X$AWTA (start local executive timing)

51.0 usec

X$ASIG (start local executive timing)

51.0 usec

X$PSIG (start local executive timing)

51.0 usec

X$PSIG (end local executive timing)

48.6 usec

X$ARD (start local executive timing)

51.0 usec

X$PRD (start local executive timing)

51.0 usec

X$PRD (end local executive timing)

48.6 usec

X$AWR (start local executive timing)

51.0 usec

X$PWR (start local executive timing)

51.0 usec

X$PWR (end local executive timing)

48.6 usec

X$ATR (start local executive timing)

51.0 usec

X$CBBR (start local executive timing)

51.0 usec

X$CBBR (end local executive timing)

48.6 usec

4.2.6.3 LEO Test Control Table

TCT PINS2 was used to gather data for the LEO tests. A description of this TCT can be found in paragraph A.2.2 of appendix A.

4.2.6.4 LEO Performance Data

During the LEO test, timing data was collected for each of the local executive routines. Timing for certain routines was dependent upon whether the applications request required a bus transmission (that is, was the request destined for another processor) or whether it affected only the local processor. The timing for the X$LXCO routine also varied. Much less time was required for X$LXCO if it was required only to queue a task of lower priority than the task currently executing. Each of these processing times was measured during the LEO test. All of the data collected for the LEO test is summarized in table 4.2-11. The first column tabulates the times measured for each routine; these measurements include the instrumentation overhead. The second column of figures tabulates the corrected measurements after the instrumentation overhead has been removed.

The "extra task" grouping represents a PINS activity controlled by the TCT which causes a low-priority task to be scheduled in the local processor and then activated by a local SIGNAL.

Executive Process	Measured Time	Corrected Time	
TRIGGER	391 us	341 us	
CANCEL	400 us	350 us	
READ	187 us	137 us	
WRITE - Other Processor	450 us	400 us	
WRITE - Same Processor	314 us	264 us	
SIGNAL - Other Processor	425 us	375 us	
SCHEDULE - Other Processor	310 us	260 us	
SCHEDULE - Same Processor	141 us	91 us	Extra Task - 523 us
SIGNAL - Same Processor	344 us	294 us	
X$LXCO - Lower Priority Task	188 us	138 us	
X$LXCO - Minor cycle Setup or Higher Priority Task	590 us	540 us	

Table 4.2-11 Local Executive Overhead Times

The times reported in the table above do not represent the total executive processing time associated with the "events" listed. Events which must be transmitted to another processor will be followed by an interrupt 3 when the transmission is completed. Requests received from another processor will cause an interrupt 3 (signifying the message has been received) and initiate a pass through the X$LXCO routine (to route the message). The total time required to service each of these events is shown in table 4.2-12 and in figures 4.2-10, 4.2-11, and 4.2-12.

Event	Base Time	Int 3 Transmit	Int 3 Receive	X$LXCO	TOTAL
Local to Processor					
READ	137 us				137 us
WRITE	264 us				264 us
SCHEDULE	91 us				91 us
CANCEL	350 us				350 us
SIGNAL	294 us				294 us
Transmitted to Other Processor					
TRIGGER	341 us	77 us			418 us
WRITE	400 us	77 us			477 us
SCHEDULE	269 us	77 us			337 us
CANCEL	350 us	77 us			427 us
SIGNAL	375 us	77 us			452 us
Received from Other Processor					
WRITE	264 us		146 us	138 us	548 us
SCHEDULE	91 us		146 us	138 us	375 us
CANCEL	350 us		146 us	138 us	634 us
SIGNAL	294 us		146 us	138 us	578 us

Table 4.2-12 Local Executive Service Times

As was done with the master executive (see paragraph 4.2.5.4) the local executive service times can be verified in a general way by comparing them to times obtained from the wait count task in the PINS program. To determine the time from the wait count, the PINS program was first run with a known processing load and the wait time was recorded. The PINS program was then run a second time with additional local executive tasks being performed and the new wait time was recorded. The difference in wait times provided the total additional processing times represented by the local executive and the applications tasks.

Two sets of events were measured in this fashion. The first set consisted of events originating in the local processor. Some of these events occurred in pairs because the design of the PINS program forces that pairing. Each of these events required some processing by applications tasks. Therefore, the times measured for them using the wait count are expected to be somewhat higher than the estimate for the local executive alone. The results of these measurements are shown in table 4.2.13. As expected, each of the times obtained from the wait count is somewhat higher than the local executive overhead time alone.

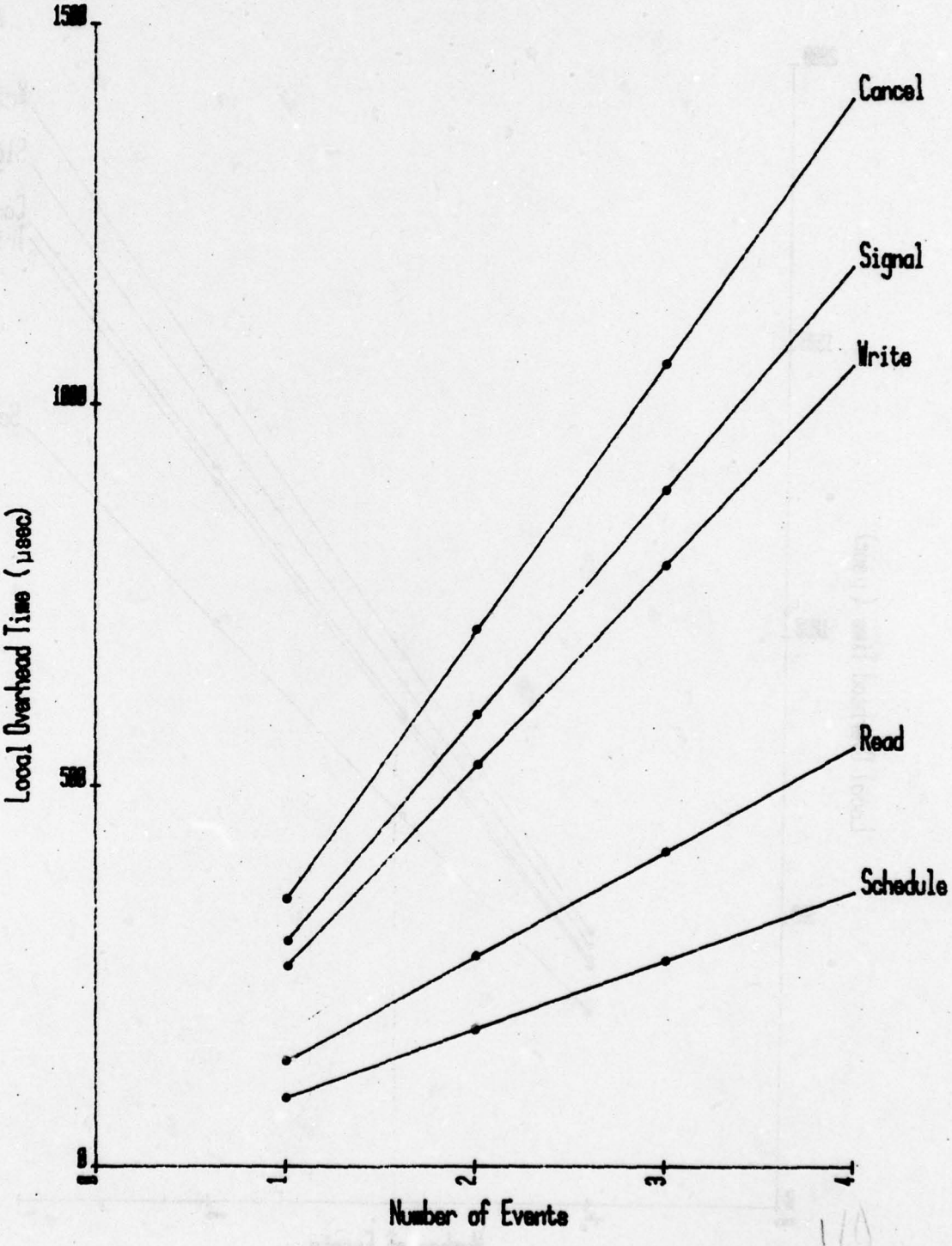


Figure 4.2-10 Local Executive Service Times - Local to Processor

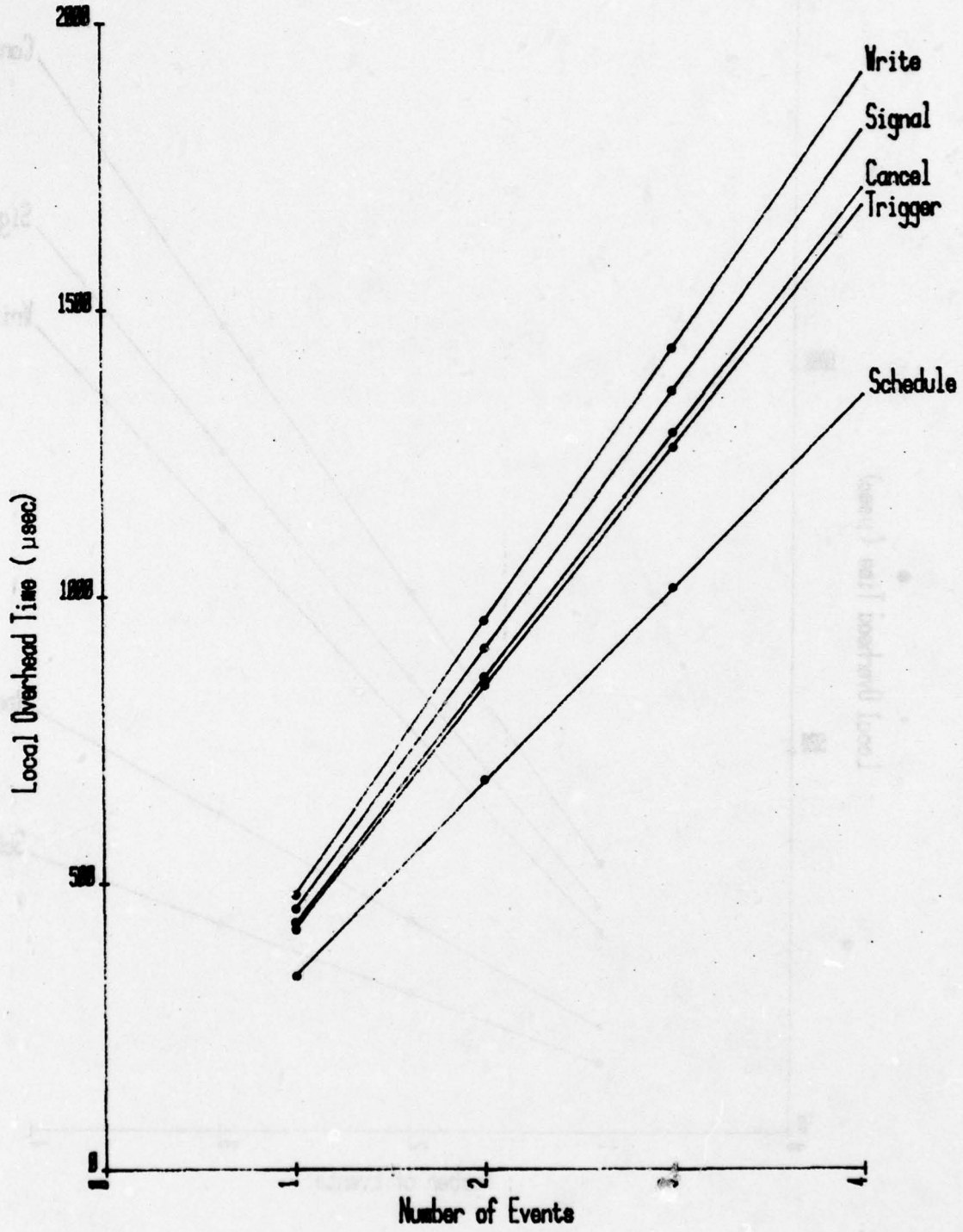


Figure 4.2-11 Local Executive Service Times - Transmitted to Other Processor

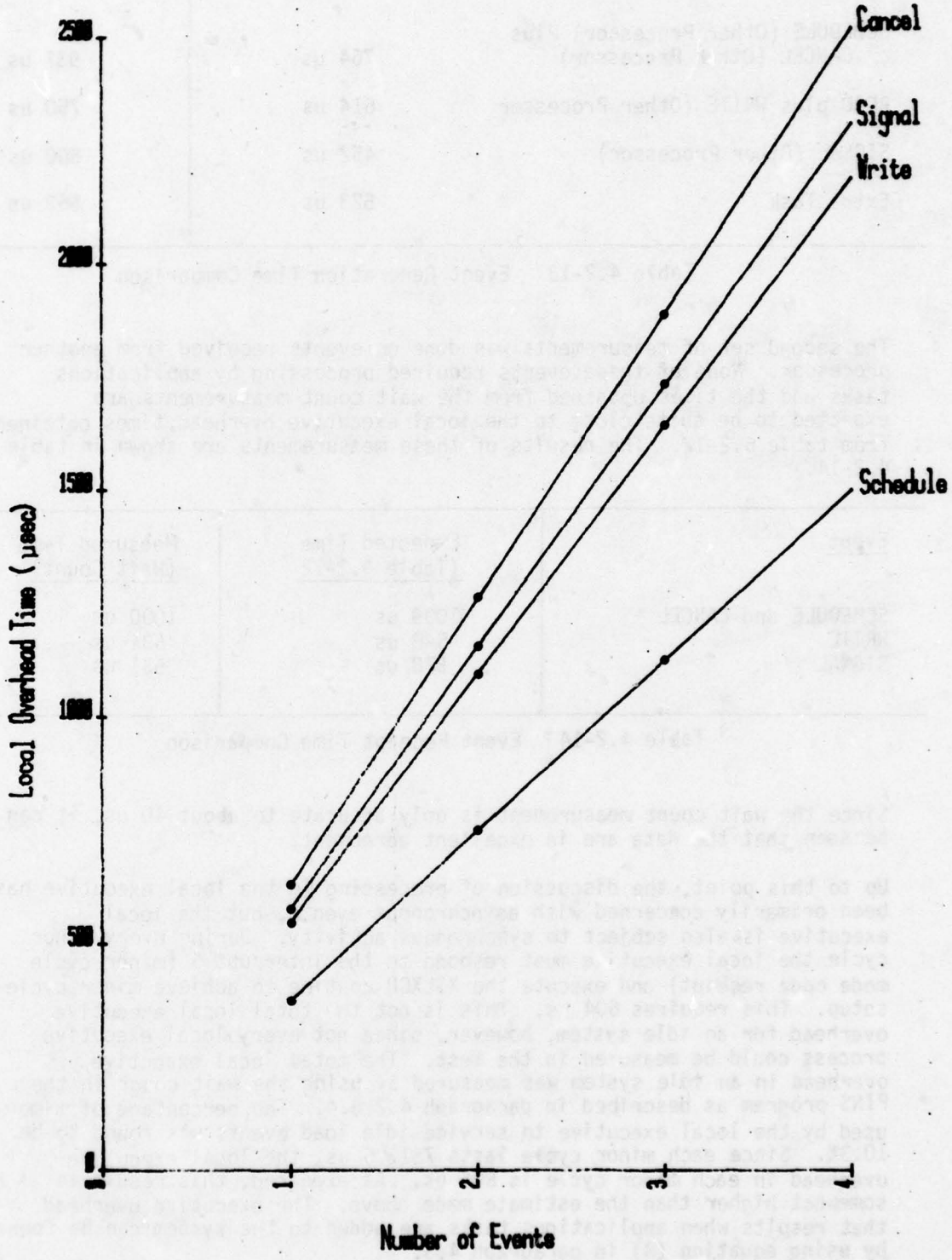


Figure 4.2-12 Local Executive Service Times - Received From Other Processor

Event	Local Executive Time	Total Processing Time
TRIGGER	418 us	500 us
SCHEDULE (Other Processor) Plus CANCEL (Other Processor)	764 us	937 us
READ plus WRITE (Other Processor	614 us	750 us
SIGNAL (Other Processor)	452 us	500 us
Extra Task	523 us	562 us

Table 4.2-13 Event Generation Time Comparison

The second set of measurements was done on events received from another processor. None of these events required processing by applications tasks and the times obtained from the wait count measurements are expected to be quite close to the local executive overhead times obtained from table 6.2-12. The results of these measurements are shown in table 4.2-14.

Event	Expected Time (Table 4.2-12	Measured Time (Wait Count)
SCHEDULE and CANCEL	1009 us	1000 us
WRITE	548 us	531 us
SIGNAL	578 us	531 us

Table 4.2-14 Event Receipt Time Comparison

Since the wait count measurement is only accurate to about 40 us, it can be seen that the data are in excellent agreement.

Up to this point, the discussion of processing in the local executive has been primarily concerned with asynchronous events, but the local executive is also subject to synchronous activity. During every minor cycle the local executive must respond to the interrupt 5 (minor cycle mode code receipt) and execute the X$LXCO routine to achieve minor cycle setup. This requires 604 us. This is not the total local executive overhead for an idle system, however, since not every local executive process could be measured in the test. The total local executive overhead in an idle system was measured by using the wait count in the PINS program as described in paragraph 4.2.5.4. The percentage of time used by the local executive to service idle load events was found to be 10.3%. Since each minor cycle lasts 7812.5 us, the local executive overhead in each minor cycle is 805 us. As expected, this result is somewhat higher than the estimate made above. The executive overhead that results when applications tasks are added to the system can be found by using equation (8) in paragraph 4.3.

4.3 Performance Data Prediction

It is possible to develop generalized predictive algorithms for DAIS executive overhead in any operational environment. The data reported in paragraph 4.2 reveal that both the master executive and the local executive are, to good approximations, independent linear systems. As such, the overhead for a multiprocessor configuration can be described as:

$$Omp = Ome + Ole \quad (1)$$

$$Orp = Ole \quad (2)$$

where:

Omp = Overhead in master processor
Orp = Overhead in remote processor
Ome = Master executive overhead
Ole = Local executive overhead

and,

$$Ome = \sum_i m_i M_i \quad (3)$$

$$Ole = \sum_j l_j L_j \quad (4)$$

where:

M_i = Processing time for "event" i in master executive
m_i = Number of occurrences of "event" i
L_j = Processing time for "event" j in local executive
l_j = Number of occurrences of "event" j

In their present form, equations (3) and (4) are of little value to the average user of the DAIS executive. The average user wants to know the overhead he can expect when performing a specific set of tasks in one minor cycle, a different set of tasks in the next minor cycle and so on. There are two areas of concern: will the overhead be excessive in any single minor cycle, and will the overall time-averaged overhead be within acceptable bounds. To satisfy the needs of this user, equations (3) and (4) can be redefined as follows:

$$Ome = K_m T + \sum_i b_i B_i \quad (5)$$

$$Ole = K_l T + \sum_j s_j S_j \quad (6)$$

where:

K_m = Percentage of processing time required by the master executive to support an "idle" load

K_l = Percentage of processing time required by the local executive to support an "idle" load

T = Duration of observed processing interval

B_i = Processing time for "event" i in master executive, where event i is in addition to idle load level

b_i = Number of occurrences of event i in measured interval

S_j = Processing time for "event" j in local executive, where event j is in addition to idle load level

s_j = Number of occurrences of event j in measured interval

As used in equations (5) and (6), "idle" load represents the smallest amount of processing that must be performed by the DAIS executives to provide an operational environment for the applications tasks. For the master executive, this includes the following processing during each minor cycle of operation:

(a) Respond to the interrupt 10 signifying the end of the last minor cycle

(b) Perform timer A processing to setup the timer to expire at the proper time in the next minor cycle

(c) Setup the DMA pointers for the synchronous instruction list to be executed during the next minor cycle

(d) Respond to an interrupt 1 and halt idle polling

(e) Respond to an interrupt 1 and start idle polling

This level of activity is required for the master executive even if there are no applications tasks in the system and is therefore referred to as the "idle" load. In the same fashion, the local executive has a minimum amount of processing which must occur every minor cycle to support an idle system. As described in paragraph 4.2, the idle load for DAIS processors operating with 128 minor cycles each second was found to be:

Master Processor - 37.5%
Local Processor - 10.3%

Substituting these values into equations (1) and (2), we find the idle load for the executives to be:

Master Executive - 27.2%
Local Executive - 10.3%

A DAIS user may now calculate overhead by counting the number of executive services being requested by applications programs, determining the time required to process these services and substituting these values into the following equations:

$$Ome = .272T + \sum_i b_i B_i \qquad (7)$$

$$Ole = .103T + \sum_j s_j S_j \qquad (8)$$

A list of the times required to support each master executive service is provided in paragraph 4.3.1, and paragraph 4.3.2 provides the corresponding information for the local executive.

4.3.1 Master Executive

Table 4.3-1 presents the complete set of master executive service times that will be needed by a DAIS user. The times are extracted from table 4.2-10 in paragraph 4.2.5.4. The values given in the right column of table 4.3-1 are the B_j of equation (7). To obtain the b_j inputs, the DAIS user must determine the total number of master executive services that will be required to support all applications tasks in all processors during the interval being studied. These services fall into the following categories:

(a) Asynchronous Global Writes - COMPOOL blocks which must be transferred over the bus asynchronously. Synchronous global writes are not counted since they are transmitted through the synchronous instruction list and require no special support from the master processor.

(b) Global Signals - Signals originating in any processor which must be sent to another processor.

(c) Schedule - Schedule a task residing in another processor.

(d) Cancel - Cancel a task residing in another processor.

(e) Trigger - Any trigger generated within the system. A trigger is the only executive service which is spread over several minor cycles. For users interested in predicting overhead on the minor cycle level, the trigger service time can be divided as follows:

    (1) Queuing Time - Master executive service provided during the minor cycle in which the TRIGGER statement is executed.

    (2) Maintenance Time - Service provided by timer A processing for every minor cycle that the TRIGGER is queued.

    (3) Transmission Time - Master executive service causing the TRIGGER COMPOOL block to be transmitted over the bus.

The service times for these phases of TRIGGER processing are listed in table 4.2-9.

After the DAIS user determines the number of master executive services that will be required in the interval T, these b_j inputs may be substituted into equation (7) to obtain the predicted overhead for the master executive.

Event	Type	Master Executive Service Time
WRITE	Local (1)	1621 us
SCHEDULE	Local	1621 us
CANCEL	Local	1621 us
SIGNAL	Local	1621 us
TRIGGER	Local	2904 us + (n-1)133 us (3)
WRITE	Non-local (2)	1402 us
SCHEDULE	Non-local	1402 us
CANCEL	Non-local	1402 us
SIGNAL	Non-local	1402 us
TRIGGER	Non-local	2685 us + (n-1)133 us (3)

(1) A type of local identifies an event that originates in the local executive resident in the master processor.

(2) A type of non-local identifies an event that originates in the local executive of a remote processor.

(3) The integer n is the number of minor cycles specified in the TRIGGER statement.

Table 4.3-1 Compiled Master Executive Service Times

4.3.2 Local Executive

The service times, S_j, for local executive routines have been reported in paragraph 4.2. These values, corrected for instrumentation overhead, are collected in table 4.3-2. This table contains the complete set of local executive service times that will be needed by a DAIS user to predict local executive overhead. The user must determine the number of times, s_j, that each service is requested during the measured interval, T, by applications tasks resident in the processor of interest. These services fall into the following categories:

(a) READ - Each COMPOOL block READ

(b) Local WRITE - Any WRITE which updates a local copy of a COMPOOL block

(c) Global WRITE - Any WRITE which must perform an asynchronous update on a COMPOOL block that is not local to the processor. Note that synchronous updates are timed as local writes.

(d) Local SIGNAL - A SIGNAL of an event which is used only in the local processor

(e) Global SIGNAL - A SIGNAL of an event used in another processor

(f) Local SCHEDULE - Scheduling a task resident in the local processor

(g) Global SCHEDULE - Scheduling a task which is resident in another processor

(h) Local CANCEL - Cancelling a task resident in the local processor

(i) Global CANCEL - Cancelling a task which is resident in another processor

(j) WAIT - Entering a WAIT state for an applications task

(k) TRIGGER - Servicing a TRIGGER request from an applications task

(l) Task Activations - Each task activated for execution in a minor cycle

After the DAIS user determines the total number of local executive services required in the interval T, these s_j inputs may be substituted into equation (8) to obtain the predicted overhead for the local executive.

Event	Type	Local Executive Service Time
READ	Local (1)	137 us
WRITE	Local	264 us
SCHEDULE	Local	91 us
CANCEL	Local	350 us
SIGNAL	Local	294 us
WRITE	Transmit (2)	477 us
SCHEDULE	Transmit	337 us
CANCEL	Transmit	427 us
SIGNAL	Transmit	452 us
WRITE	Receive (3)	548 us
SCHEDULE	Receive	375 us
CANCEL	Receive	634 us
SIGNAL	Receive	578 us
TRIGGER		418 us
Event Wait		303 us
Task Activation		74 us

Notes:

(1) "Local" identifies an event that is originated and used by the same processor - the master executive is not involved.

(2) "transmit" identifies an asynchronous request which must be sent to the master executive for servicing (a bus transmission is involved).

(3) "receive" identifies an asynchronous request which originated in another processor and was sent to this processor by the master executive via a bus transmission.

Table 4.3-2 Compiled Local Executive Service Times

4.4 Phase II Summary

The validity of the predictive algorithms developed in paragraph 4.3 may be verified by applying them to the DINS program developed for Phase II. As the first step in determining executive overhead, the total number of executive services requested for DINS must be determined. The DINS processing load is shown in figures 3.2-5 and 3.2-6. Nearly every DINS service request is local to the remote processor. All of the DINS tasks that are active during a test reside in the remote processor. The only executive service request made by DINS which causes an asynchronous bus transmission is an asynchronous WRITE.

A study of figures 3.2-5 and 3.2-6 shows few differences in service requests in the first and succeeding seconds of a Kalman filter sequence. In order to eliminate redundant calculations, the predictive algorithms will be applied only to the DINS load depicted in figure 3.2-6. This processing load occurs in the DINS system during five out of every six seconds.

By summing the executive requests listed for every minor cycle shown in figure 3.2-6, we obtain the total number of requests that are made in a one second period. This data is shown in table 4.4-1. This table defines all of the executive service requests originated by the applications tasks in the remote processor. The values supplied in this table constitute the s_j of equation (8) in paragraph 4.3. The S_j inputs are found in table 4.3-2. The processing times listed in this table were shown to be valid for the DINS system by instrumenting the DAIS executive while DINS was running; the service times observed for DINS were the same as those obtained for PINS. By substituting the s_j values into equation (8), we obtain the following:

$$O_{1e} = .103\ (10^6\ us) + 723\ (137\ us) + 304\ (264\ us) + 32\ (477\ us) + 15\ (294\ us) + 8\ (91\ us) + 276\ (74\ us)$$

$$= 103\ ms + 99.1\ ms + 80.3\ ms + 15.3\ ms + 4.4\ ms + 0.7\ ms + 20.4\ ms$$

$$O_{1e} = 323.2\ ms \text{ over a one second period}$$

	0	1	2	3	4	5	6	7	TOTAL
READ		3	416		192		112		723
WRITE-Local		48	128		80		48		304
WRITE-Global			32						32
SIGNAL								15	15
SCHEDULE							8		8
Task Activation		34	144		64		17	17	276

Table 4.4-1 Anticipated DINS Executive Service Requests for One Second Period

We can also determine the predicted overhead for the local executive in the master processor. In the DINS program, the local executive has only one applications function - to receive the asynchronous COMPOOL blocks sent to it by the remote processor. Using the number of global writes listed in table 4.4-1, we can determine that the local executive in the master processor must receive 32 COMPOOL blocks each second. From table 4.3-2, we can determine that each of these will require 548 us of processing time. Substituting these values into equation (8) we obtain:

$$O_{1e} = .103\ (10\ \ us) + 32(548\ us)$$
$$= 103\ ms + 17.5\ ms$$

$$O_{1e} = 120.5\ ms \text{ over a one second period}$$

Finally, we can predict the processing load in the master executive. As pointed out above, the only master executive service required in the DINS program is to field the asynchronous COMPOOL blocks written 32 times each second. By substituting the service time from table 6.3-1 into equation (7) of paragraph 4.3, we obtain:

$$O_{me} = .272\ (10^6\ us) + 32(1402\ us)$$
$$= 272\ ms + 44.9\ ms$$

$$O_{me} = 316.9\ ms \text{ over a one second period}$$

Substituting O_{me} and O_{le} into equations (1) and (2) of paragraph 4.3, we determine the predicted overhead in each processor:

O_{mp} = 437.4 ms over a one second period = 43.7%
O_{rp} = 120.5 ms over a one second period = 12.0%

The predictive algorithms should always be applied to processing intervals that contain an above-average number of executive service requests. Careful analysis of such intervals will enable a designer to eliminate possible "choke points" in a system. Choke points occur when the system processing load exceeds the resources available. When such a situation arises, the execution of tasks will be delayed one or more minor cycles until the backlog can be eliminated. Table 4.4-1 indicates that minor cycle 2 will have the heaviest DINS processing load, so minor cycle 2 is the logical place to begin a search for potential choke points.

Overhead for minor cycle 2 is calculated in the same fashion that the overhead was predicted for the full one second interval. Table 4.4-1 lists the number of executive services required in minor cycle 2 for a one second period (note that minor cycle 2 repeats 16 times in a one second interval). By substituting these values and the service times from table 4.3-2 into equation (8), we obtain:

$$O_{le} = .103(125000\ us) + 416(137\ us) + 128(264\ us) + 32(477\ us) + 144(174\ us)$$
$$= 12.9\ ms + 57.0\ ms + 33.8\ ms + 15.3\ ms + 10.7\ ms$$

$$O_{le} = 129.7\ ms \text{ in a } 125\ ms \text{ period}$$

Since the predicted overhead alone exceeds the processing time available, it is apparent that minor cycle 2 will be a DINS choke point. A major portion of the processing planned for minor cycle 2 will not be completed in that minor cycle. Fortunately, minor cycle 3 has a very low processing load and will be able to accommodate the tasks that cannot be completed in minor cycle 2.

Since the most heavily loaded DINS minor cycle was found to be a choke point, we must now look at the next most heavily loaded minor cycle. A study of table 4.4-1 shows that minor cycle 4 will have the second highest overhead in DINS. We calculate this overhead as we have done before, by substituting the appropriate values into equation (8):

$$O_{le} = .103(125000\ us) + 192(137\ us) + 80(264\ us) + 64(74\ us)$$
$$= 12.9\ ms + 26.3\ ms + 21.1\ ms + 4.7\ ms$$

$$O_{le} = 65.0\ ms \text{ in a } 125\ ms \text{ period}$$

Although the overhead load in this minor cycle is not as critical as in minor cycle 2, it is still severe. In order to service the requests generated by the applications tasks in minor cycle 4, the local executive will require more than 50% of the available processing time. This will severely limit the number of applications tasks that can be handled in this minor cycle. As a rule of thumb, few systems perform acceptably when the executive overhead (in the absence of applications requests) exceeds 30% of the total processing load; using this guideline, it is reasonable to assume that minor cycle 4 could also be a DINS choke point.

The isolation of these potential overload points in DINS demonstrates the value of the predictive algorithms. In the normal system design process, the designer would now go back and restructure the task load to eliminate these trouble spots. This was not possible with DINS, however, since the DINS design was already complete and the system was being tested in the laboratory when the predictive algorithms became available. Thus, the measurements made for Phase II should show evidence of overloading in minor cycle 2 and perhaps also in minor cycle 4.

For Phase II, DINS was instrumented and tests were run to determine the performance of the DAIS executive. For the most part, these tests were successful; however, the high level of DINS activity during minor cycles 2 and 4 caused so many entrances to executive routines that the ability of our equipment to record the data was exceeded. Hence, complete information was never collected for minor cycles 2 and 4. Partial data on these minor cycles was obtained by reducing the number of instrumentation hooks being used. Even so, some of the DINS READs and WRITEs were never successfully trapped and no data could be gathered on the number of task activations. The events that were trapped are shown in table 4.4-2. Notice that the number of events trapped for low frequency executive requests (i.e., global WRITE, SIGNAL, and SCHEDULE) is the same as the number expected in table 4.4-1. This indicates that the DINS program was performing as designed and discrepancies in the number of high frequency events are due entirely to the limited capacity of the recording equipment.

	MINOR CYCLES								
	0	1	2*	3	4*	5	6	7	TOTAL
READ		3	160	112			112		387
WRITE - Local		48	64		80		48		240
WRITE - Global			32						32
SIGNAL								15	15
SCHEDULE							8		8

* Data recording equipment overloaded in minor cycles 2 and 4

Table 4.4-2 Observed DINS Processing Load for One Second Period

The data in table 4.4-2 permit some interesting observations about the DINS choke points that were predicted earlier. By comparing the number of READs observed in table 4.4-2 with the number expected in table 4.4-1, it is apparent that not all of the tasks scheduled for activation in minor cycle 2 are in fact being activated in that minor cycle. Many of the READs we expected in minor cycle 2 do not get executed until minor cycle 3. Our prediction of a system overload in minor cycle 2 is thus borne out by measurements in the laboratory. The value of using the predictive algorithms to locate potential choke points in a new system has been graphically demonstrated.

The primary purpose of the predictive algorithms is still to predict the overhead for a system with a known processing load. To determine the accuarcy of the predictive algorithms, the total DAIS executive overhead for DINS was measured in the master processor and determined to be 41.6%. By comparing this result to the 43.7% figure predicted earlier in this paragraph, we find that the predictive algorithms came within 2% of the actual executive overhead. The verification phase has thus demonstrated that the predictive algorithms developed in this study can provide other users of the DAIS executive with an effective means of predicting overhead for new systems.

5.0 DAIS Executive Support Analysis

During the development of the Pseudo-Integrated Navigation System (PINS) and the DARTS Integrated Navigation System (DINS), the DAIS Software Development Standards (SDS) and the DAIS Support Software were evaluated by actually using them and evaluating the results through observation. The analysis of the DAIS Software Development Standards (PA 200101) appears in paragraph 5.1. The analysis of the DAIS Support Software appears in paragraph 5.2. All comments made in this section refer to the 1977 version of the DAIS executive, its support software and the accompanying documentation. AFAL has been and is continually improving this software and its documentation. Many of the problems noted in this section have already been corrected in newer versions of the executive.

5.1 DAIS Software Development Standards

The DAIS Software Development Standards, documented in PA 200101 (15 Jan 76 Draft), provide guidelines for building DAIS applications software systems. The DAIS Software Development Standards are comprised of:

- o Architecture Standard
- o Documentation Standard
- o Structured Requirements Design Standard
- o Program Specification Design Standard
- o Implementation and Verification Standard

Paragraph 5.1.1 provides a general analysis of each of these standards. Paragraph 5.1.2 discusses in detail problems encountered with the use of the standards, errors in the documentation, and specific recommendations for changes.

Two avionic software systems were built during the course of this study using the standards: the DARTS Integrated Navigation System (DINS) and the Pseudo-Integrated Navigation System (PINS). These guidelines were followed during the development of each system.

In general, the standards were found to be adequate in describing methodology for building dependable and maintainable DAIS appliations software systems. The standards were successful in reducing program complexity by imposing a hierarchical control structure, top-down design, and a high degree of modularity.

5.1.1 Analysis of DAIS Software Development Standards

5.1.1.1 Architecture Standard Analysis

The DAIS architecture standard describes the system structure and system control structure as consisting of a MIL-STD-1553A/DAIS multiplex bus with DAIS processors and DAIS Remote Terminals (RT) communicating via this bus. This architecture standard also defines the relationship between the applications software and the hardware system.

The major strength identified in this standard was the functional isolation of components in the system architecture. This functional isolation stressed the individual development of functional components all built to a rigid interface which assures the orderly integration of these components into an operational system.

Another major strength relative to the standard is the requirement for absolute isolation of the operating system or executive computer program from the applications software. This functional separation forces adherence to interfaces during the applications software development (which is unusual in avionics programs), because the executive computer program was complete and under configuration control.

5.1.1.2 Documentation Standard Analysis

The documentation standard addresses the structured representation of a program module, the definition of a program module's data module and naming conventions.

The documentation standards specified in the DAIS SDS were not applied to either program used in this evaluation. These standards follow general industry practice, therefore existing Boeing standards were used.

One specification in these standards which departed from general industry practice was determined to be quite good and was integrated into existing standards. This was the specification referred to in the SDS as a "data module." The data module defines the realtime interface of a program module with the rest of the operational system, including the local executive.

5.1.1.3 Structured Requirements Design Standard Analysis

This standard addresses the specification of mission requirements in terms of a structured design. The benefits that are anticipated through this methodology include data integrity, and a hierarchical relationship of functions.

The standard specifies a hierarchical top-down methodology of designing a control structure. This methodology has been proven throughout industry and was judged to be a good standard for system design.

The standards implied that each program module should perform a "single function". This interpretation of the standards was used to design the DINS program and was found to generate a plethora of task modules. The standards should be reworded so that it is clear that closely related program modules may be integrated into an executable entity.

5.1.1.4 Program Specification Design Standards Analysis

This section of the DAIS SDS specifies the methodology for the construction or coding of program modules. Included in this methodology are detailed naming conventions and usage of the JOVIAL J73/I high order language. Also specified is the realtime interface definition with the DAIS executive, a system generation methodology including a "building block" program structure, and a set of rules for the use of tasks, task states and events.

The key element of this section is the specification of the executive/applications interface. The rigid definition of this interface is the one single aspect of these standards which is most important to the overall development of applications software. The overall integration effort was greatly reduced because absolutely no executive computer program modification or development was necessary.

Naming conventions were also defined in these standards and were judged to be of relatively little benefit. In many cases, the effort to develop meaningful or mnemonic names produced such lengthy results that confusion was actually increased.

The interface with the executive computer program is maintained through a series of realtime statements. Because of their importance, these statements should be documented in more detail and include more examples. This is particularly true of the SCHEDULE realtime statement.

5.1.1.5 Implementation and Verification Standard Analysis

These standards address the testing and integration testing of the applications program after the program modules have been coded. The standard addresses two testing concepts. One was unit test of the program module which is a very necessary and important test. The other is the actual integration of the unit-tested program modules. The standard specified that this be accomplished in a top-down manner which has been proven throughout industry.

5.1.2 Problems Encountered With Standards

In using the standards defined in PA200101 (15 Jan 76 draft) some areas were discovered where the standards were somewhat ambiguous or unclear. This section describes these problems or deficiencies.

5.1.2.1 Architecture Standard Problems

For this study one of the guidelines was to live within the architecture standards. Since these standards specify interfaces beyond the scope of executive control, no particular deficiencies were discovered. For a federated control system operating in a single bus architecture comprised of DAIS processors and DAIS RTs, these standards are very applicable.

5.1.2.2 Documentation Standard Problems

As was reported in paragraph 5.1.1.2 above, these standards were not used in this study; however, the standards were reviewed. One discrepancy apparently exists between the structured requirements design standard and the program specification design standard. The discrepancy is found in paragraph 3.2.2.9 where "outer subroutines invoked" and "outer data variables referenced and assigned" are discussed. According to the structured requirements design standard and the program specification design standard these types of references or definitions are prohibited.

5.1.2.3 Structured Requirements Design Standard Problems

Some problems were noted in this standard and some areas needed further clarification for the designer. These problems are discussed below.

5.1.2.3.1 Controllers Remaining Active

Paragraph 3.3.3-5 states that a task should not suspend itself by entering a WAIT state, waiting for an event signal. Paragraph 3.3.3-6 states that a controller should remain active to field error conditions from any of its descendants. Paragraph 3.3.6 I.B.2 states that the priority of a process is higher than the priority of all processes below it in the control tree. Taken together, these three standards present the designer with a difficult problem. A controller must remain active whenever one of its descendants is active, but it is not allowed to enter a WAIT state to do so. Since the controller will generally have a higher priority than any of its descendants, the controller will execute indefinitely, precluding the execution of its descendants. Either paragraph 3.3.3-5 or 3.3.3-6 of the standard need revision to correct this difficulty.

5.1.2.3.2 Data Integrity

Paragraph 3.3.6 (11) contains a statement that a module may not receive data nor assign data unless the data is "known" by its controller. This statement creates problems since it implies that any data read by a process must be written by its controller or if a process writes data it must be read by its controller. Implementation, however, restricts multiple access of the same data block. Therefore, the intent of this statement should be clarified in the document.

5.1.2.3.3 Data Access

Paragraph 3.3.6 (12) states that communications among processes must use COMPOOL blocks transferred through the executive system. This restriction ensures data integrity but it places unnecessary burdens on the system in general.

This large number of executive requests could be reduced substantially if only the controller were required to access the COMPOOL blocks used by its descendants.

As an example, consider the situation where a controller has a set of 10 tasks operating under it. As a group, these tasks require four input COMPOOL blocks, and the average task READs two of these and WRITEs another. As a group, these tasks generate four output blocks which will be synchronously written onto the data bus. Thus, as a unit, the group READs four input blocks and WRITEs four others. As individual tasks, 20 READs and 10 WRITEs are required. This task group executes 16 times a second which means that 480 requests for executive service will be made in a one second period.

If the data access restriction were not imposed, a COMPOOL could be defined which is known to the controller and its descendants. Data could be READ by the controller using the executive services and moved to this COMPOOL. Since the descendants have implicitly READ the data (because the controller has), they could manipulate the data without requiring any further support from the DAIS executive. When all of the descendants have completed processing, the controller could WRITE the data produced by its descendants, placing it in the executive controlled database using executive services. This procedure would not only reduce system overhead but would satisfy the intent of paragraph 3.3.6 (11) discussed above.

This approach could be applied to our previous example by defining a local COMPOOL available to all 10 tasks. The controller could READ the four input blocks into this COMPOOL, wait for the descendants to finish execution and then WRITE the output COMPOOL blocks. Data integrity would be maintained since only the controller and its descendants would have access to the data and these tasks would execute in a predefined sequence (in order of task priority). For our previous example, only 128 executive service requests would be generated by the controller. Adoption of this approach would save 354 executive calls in a one second period.

5.1.2.4 Program Specification Design Standard Problems

Overall these standards provide useful guidelines for coding applications tasks operating in a DAIS environment. There are some deficiencies which are outlined below.

5.1.2.4.1 Task State Event

Paragraph 3.4.5.2.1 of the DAIS SDS discusses the use of a task state event in the condition set of a SCHEDULE realtime statement; however the DAIS local executive does not support this option.

A change in one task's state cannot change another task's state from INACTIVE to ACTIVE, a situation which was only discovered only after a program was designed, built, and undergoing laboratory testing. Finding this problem and developing another method of controlling activation required several days of debugging and rebuilding effort. WAIT statements were finally used, which is in violation of paragraph 3.3.3-5 of the Standards.

5.1.2.4.2 COMPOOL Block Defintions

Paragraph 3.4.4.3.1 of the SDS restricts the length of a COMPOOL block to a maximum of 32 words. This restriction is imposed by MIL-STD-1553A/DAIS protocol since the bus interface hardware allows only 5-bits for data length in a bus message. This is a reasonable standard for external data which is transmitted on the data bus; it is even reasonable for data passed between logical branches of the tree structure; however in actual implementation, very few tables internal to a control structure lend themselves to 32-word block definition. Further, data structures requiring two or three levels of nesting are impossible to define in 32-word blocks with intervening tag words; J73/I does not support processing of such a data structure.

Even for external and intertask data transfers, the data need not necessarily be restricted to 32 words since the executive computer program could easily accomodate blocks of varying length and break them into the 32 word blocks required for transmission. Means of providing larger block sizes (largely for internal use by tasks reporting to a single controller) should be investigated for future versions of the DAIS system..

(This discussion relates to the discussion in 5.1.2.3.3 above.)

5.1.2.4.3 Task Sizing

Several guidelines state that the size of program units should be kept between 10 and 40 executable statements. In addition, there is an implication that program units are equivalent to executable tasks. As a result, the system designer is discouraged from combining a number of program modules into executable tasks. If the system design calls for a controller with three different sets of calculations to perform, the standards imply that the designer define four separate executable tasks - the controller and three calculators. This is an inefficient implementation guideline which may degrade system performance.

For example, consider the situation described in paragraph 3.2.3.3 where many different COMPOOL blocks are accessed. If the three calculators were included in the controller as imbedded procedures, there would be no need to define additional COMPOOL blocks to satisfy paragraph 3.3.6 (12) since the data would already be available to the controller as stated in 3.3.6 (11). None of the benefits of small program units would be lost since the program modules would be maintained as imbedded procedures.

5.1.2.4.4 TRIGGER Statement Parameters

The TRIGGER statement contains two time fields, time and delta time. At the end of the SDS TRIGGER description (paragraph 3.4.3.5.2.8, p.55) is a statement that the time field alone is sufficient for some applications. This statement implies that the delta time field may be omitted from the TRIGGER statement if the time field provides sufficient accuracy. An attempt to use the TRIGGER statement without a delta time field proved unsuccessful; the delta time field must be included in every TRIGGER statement. The documentation should be clarified in this respect.

The SDS states that the delta time field is scaled in units of ten microseconds (paragraph 3.4.3.5.2.8, p.55). This is incorrect. The delta time field is entered as an integral number of milliseconds. The documentation needs to be corrected on this point. In addition, the documentation should indicate that weapon releases cannot achieve relative spacings of more than one millisecond accuracy.

5.1.2.4.5 COMPOOL Directive

The SDS shows two examples of the COMPOOL directive in table 3.4.9 on page 62. Both examples incorrectly show the format to be !COMPOOL 'XXXXXXX.CMP' . Attempts to use this format proved unsuccessful. The correct format is !COMPOOL ('XXXXXX.CMP'); .

5.1.2.4.6 EJECT Directive

The SDS has two tables (table 3.4-11, p.66 and table 3.4-13, p.69) which demonstrate the use of the EJECT directive. Both examples incorrectly show the format to be EJECT . Attempts to use this format were unsuccessful. The correct format is !EJECT; .

5.1.2.4.7 Priority Field in SCHEDULE Statement

Paragraph 3.4.3.5.2.1 discusses the PRIORITY field in the SCHEDULE statement but omits the option of setting PRIORITY = PRIVILEGED. Since privileged tasks have many important uses in the DAIS software system, the standards should be revised to discuss this option.

5.2 DAIS Support Software

Various DAIS Support Software tools were used during the development of two DAIS application software systems. This support software was found to be generally adequate in enabling straightforward and dependable DAIS application system development. Section 5.2.1 provides general analysis of each program of the DAIS Support Software which was used. Section 5.2.2 discusses in detail the problems encountered with each program and makes specific recommendations for changes. Section 5.2.3 details the problems encountered with each program's documentation and recommends changes.

5.2.1 Analysis of DAIS Support Software

5.2.1.1 JOVIAL J73/I Compiler Analysis

The compiler reads programs coded in JOVIAL J73/I and produces relocatable files in AN/AYK-15 machine language. This program was found to be fairly efficient and quite reliable. The Boeing engineers who used the compiler varied from novice to experienced programmers, but none had previous JOVIAL J73/I experience. All were producing source code at the end of a week and the compiler generated reliable machine language code from these source programs.

The first version of the compiler, delivered as GFE, did have some errors in the code generation. These errors were predictable and could be worked around with a great deal of time-consuming effort. The Air Force Avionics Laboratory provided an updated compiler which corrected these problems.

Other problems resulted from weak documentation. For example, the JOVIAL J73/I Computer Programming Manual (MA204200-1, Oct. 75) provides relatively meaningless examples which appear to be directed more toward students than to software designers and coders.

One other problem concerned the error/diagnostic messages generated by the compiler. These messages were not always clear and they were inadequately described in the manual.

5.2.1.2 PALEFAC Preprocessor Analysis

The PALEFAC preprocessor (PP) program reads applications software programs written in JOVIAL J73/I. One function of the program is to extract information relative to the applications program interface with the DAIS local executive. The program reformats this information and writes a file used by the PALEFAC program. A second function of this program is to examine the J73/I source code for compliance with DAIS Software Development Standards.

The PALEFAC PP successfully read JOVIAL J73/I source code programs and produced the input file for the PALEFAC program. In addition, the PALEFAC PP audited the J73/I source code for compliance with most of the DAIS Software Development Standards.

The cost (in computer resources) of operating the PALEFAC PP program was extremely high. The cost to run this one program was found to exceed the total cost of the rest of the system generation process.

5.2.1.3 PALEFAC Analysis

The PALEFAC program reads the file created by the PP program along with system configuration data and produces the tables that provide and maintain the interface between the executive and the applications. The output of this program is a series of JOVIAL J73/I source programs which contain the tables for task control and event control, synchronous bus lists to provide synchronous bus traffic, and global COMPOOL blocks for the data which is transferred either intertask or via the bus. The tables used for the asynchronous data transmission are also produced.

Another output of the PALEFAC program is the control statement for the software test stand linker (LINKS). It was found that PALEFAC does not produce control statements for the Hot Bench Computer (HBC) or AN/AYK-15 processor load module.

The tables produced by PALEFAC served their purpose and were generally complete. Some errors were discovered in the program but new versions of the program supplied by AFAL corrected most of the problems.

5.2.1.4 Software Test Stand Linker (LINKS) Analysis

The LINKS program was used to link-edit the relocatable modules produced by the JOVIAL J73/I compiler and the ALAP assembler program. No errors were discovered in the operation of the program.

LINKS documentation, however, provides too little information to be of use to an engineer. The linkage control necessary to build loadable modules was provided by AFAL personnel, so Boeing engineers did not have to attempt to interpret the document and construct linkage control.

5.2.1.5 Performance Monitor and Control (PMC) Analysis

The PMC program is a realtime software control and monitoring tool. This program was used in a very limited way during this study, therefore a detailed evaluation of this program was not made.

5.2.1.6 DAIS Diagnostic Program (M$DIAGN) Analysis

This program is provided with the executive software for the purpose of displaying elements of the executive database in non-realtime. The program was used during system integration debugging and was found to be fairly useful. Better use might have been made of M$DIAGN had documentation been available.

5.2.1.7 DAIS Processor/Cross Assembler (ALAP) Analysis

ALAP reads assembler language code specified for the AN/AYK-15 processors and produces machine language programs for the AN/AYK-15 processor. The output of this program was suitable for processing by the STS linker program LINKS.

5.2.1.8 ASYTRN Analysis

ASYTRN reads the load file created by the linkage editor program and produces a file in the proper format to be loaded by the CIU connected to the AN/AYK-15 processors. The program performed this function without any problems.

5.2.2 Problems Encountered With DAIS Support Software

5.2.2.1 J73/I Compiler Problems

5.2.2.1.1 Padding Word Insertion

The entire database of a DAIS application software system is composed of small groupings of data called blocks. These blocks are centrally defined in a COMPOOL for reference by all program units. Blocks are the data structures used to pass information between tasks within the same processor and between processors and remote terminals.

The J73/I compiler is generally free to re-order items within a block or to insert padding words between items in order to efficiently assign memory locations. This occurs frequently with floating point items, which must be assigned to an even memory address for the most efficient code generation. The problem encountered here is with blocks used to transfer data to and from remote terminals. In this case, the remote terminal design may require that a certain structure be maintained for a block, but the designer has no simple, straightforward method (such as a "keyword") of telling the compiler to leave the block exactly as the designer has specified. The use of the Jovial overlay declaration will prevent the compiler from re-ordering items in a block, but it does not prevent the compiler from inserting padding words.

Additionally, the compiler gives no indication to the programmer that a re-ordering or padding word insertion has taken place for a given block. In our case, this caused interface problems between the DAIS processors and the Harris/6 computer which were not discovered until the system had been built and was executing. This could easily have been avoided had the compiler issued warning messages or if the programmmer had a clean method of preventing the compiler from reordering or padding COMPOOL blocks.

5.2.2.1.2 Incorrect Block References

During the first stages of DINS development, a version of the J73/I compiler dated September 28, 1977 was used. This compiler was found to generate incorrect address references for some COMPOOL blocks. In some cases the references were one or two words off, but in another case they were consistently off by 32 words. The compiler generated no diagnostic messages for these cases, so the problem was discovered only after the program was executing on the DAIS processors. Since the cause of the problem was not immediately understood, lengthy machine code patches were generated so that testing could proceed. Eventually source code changes were discovered which enabled the compiler to generate correct references. This problem did not occur with the compiler dated August 31, 1978, which was used for the later DINS system builds.

5.2.2.2 PALEFAC Preprocessor Problems

The PALEFAC Preprocessor reads the DAIS applications software source code and produces a file of information needed by the PALEFAC program. Every applications task must be processed by the preprocessor, and certain changes to a program unit in a task require that the task again be processed by the preprocessor before another PALEFAC run is made. The preprocessor builds a disk file called the PALEFAC Module Input (PMI) file which is then read by the PALEFAC program.

5.2.2.2.1 Cost to Execute Preprocessor

The preprocessor proved to be a very expensive program to execute. A major change to a DAIS applications software system, as routinely occurs when using top-down integration techniques, typically requires a preprocessor run involving a large share of the tasks in the system. In this case, the computing cost for the preprocessor run was approximately twice that of the balance of the system generation (which includes a PALEFAC run and a LINKS run). This cost factor inhibits the designer from making frequent system rebuilds, but this rebuild capability is one of the strongest assets of DAIS and should not be eroded by excessive computing costs. It is recommended that the preprocessor be examined for possible areas for optimization.

5.2.2.2.2 Lack of User Status Information

More than once during the study, a preprocessor run executed on the University of Washington DEC-10 computer required more than four hours to complete. During this period, the user has no feedback at the terminal which will allow him to determine if the run is progressing normally. Since the user does not wish to stop a run prematurely, the tendency is to let the program execute. Nearly every time that the preprocessor program was run, the computer operator at the University of Washington called to tell us that the program appeared to be in a loop. On one occasion the operator called three times before the run was completed. Although the preprocessor generally performed well, there were occasions when the program did begin looping. Since no status information was provided, the runs were allowed to execute until it became obvious that they were not going to finish. These runs which had to be redone cost approximately $1000 each in unnecessary computer rental time.

A file which contains status information, the PALEFAC Preprocessor Text Output (PPT) File, is not made available to the terminal user until the end of the run. It is recommended that the program be modified to provide at least a reasonable subset of the PPT file to the terminal user. This information should include, at the least, the input file being operated upon and program phase of the PALEFAC preprocessor program which is currently operating.

5.2.2.2.3 Truncation of Input

The preprocessor requires that all input must be in the form of card images. The preprocessor documentation does not adequately describe this requirement. During the study, the preprocessor was run with input from a disk file which had records exceeding 80 characters in length for input. The program appeared to achieve a normal completion. It was not until PALEFAC was subsequently run using the PMI file produced by the PP that discrepancies were noted in the output. Upon investigation, it became apparent that certain entries in the PMI file did not contain all of the expected data. Further investigation showed that all of the missing data should have been extracted from the last lines in a file, and the cutoff point coincided with an input line that exceeded 80 characters in length. After the offending line was shortened, the PMI was updated using the preprocessor and PALEFAC executed normally.

It may be deduced that an input card image with more than 80 characters causes the preprocessor to terminate processing on that file, although the date/time group is still appended to the truncated PMI entry. Since no warning message is presented to the user, PALEFAC was executed before the problem was found. Several engineering hours were required to isolate the problem, which should and could have been reported by the program itself. This problem may have been prevented had the documentation adequately described the requirement.

5.2.2.2.4 Compliance with Software Development Standards

The preprocessor is designed to examine the applications source code to determine compliance with the DAIS Software Development Standards. In general, the preprocessor does an excellent job of monitoring the code for compliance with the standards. Unfortunately, all standards are not checked by the preprocessor and erroneous code can slip through. Three such items found during this study are:

a. The SCHEDULE statement is used to sequence and provide conditions for the execution of tasks. One of the fields of the SCHEDULE statement which the designer is required to supply according to the standards is the priority field, which is used to determine each task's priority in relation to all other tasks in the system; however, the preprocessor does not test to see that the priority field has been supplied. If the priority field is inadvertently omitted, the task being scheduled is assigned a privileged priority status, which is for a special high-priority class of tasks. No error or warning message is produced by the preprocessor and the user may not be aware of the problem until debugging runs in the laboratory demonstrate an undesired task execution sequence.

b. Another field in the SCHEDULE statement is the period specifier, which indicates how often the task being scheduled is to be executed. Because of the DAIS executive design, this period specifier is limited to values which are powers of 2; however, the preprocessor does not check the value supplied for the period field of the SCHEDULE statement. During the DINS development, a period was incorrectly coded as 127. The preprocessor produced no diagnostic message, and the end result was an error in the task execution sequence which was discovered in the laboratory.

c. For each COMPOOL block referenced by a task, the standards require that the designer indicate the type of the block as READ (referenced only in a READ statement), WRITE (referenced only in a WRITE statement), or UPDATE (referenced in both READ and WRITE statements). The preprocessor however, does not check to see that a block declared as UPDATE is in fact both read and written by the task. In one instance during the DINS development, a task had a COMPOOL block declared as being UPDATE, but the READ statement for the block was inadvertently omitted. The preprocessor produced no error or warning message, and the error had to be discovered in the laboratory.

It is recommended that the preprocessor's capability in monitoring for adherence to the Software Development Standards be improved.

5.2.2.3 PALEFAC Problems

This program reads the preprocessor's output (the PMI file) as well as two other designer-coded files and produces several database modules for use by the DAIS executive. PALEFAC is generally run once for each DAIS applications system build. If an error is found in a task, it is corrected in the source code and the task is recompiled. It may then require a second pass through the preprocessor, and finally PALEFAC is run again. When the run is error-free, the designer can compile the PALEFAC output modules and then proceed to the link step.

5.2.2.3.1 CLASS Field Processing

One of the PALEFAC input files that the designer codes is called the PALEFAC Global Input (PGI) file. In one section of this file the designer explicitly defines all data communications over the multiplex bus which are to take place on a regular, repeated basis. For each of these synchronous messages defined, the designer can specify the CLASS of retry he wants the system to attempt in the event of a failure of the message transmission. If he chooses a CLASS called "automatic retry," he can enter an additional field which specifies how many times (0-3) that he wants the failed transmission to be retried.

In the DINS application, we attempted to use this retry field with no success. No matter what retry count we coded, PALEFAC did not produce any non-zero retry count in its output. We also attempted to let PALEFAC produce a default retry count for each message by omitting the CLASS field, but this was not successful.

This PALEFAC error cost us much time and effort. A non-zero retry count must be specified for each message in order to enable the system to recover from minor bus transmission problems, so each time the DAIS processors were loaded, it was necessary to enter patches to the entire bus message list.

5.2.2.3.2 Required Pairing of Transmissions with Receptions

One of the things the PALEFAC program checks is the COMPOOL block READ and WRITE statements included in each applications task. If a particular block is written by any task, PALEFAC checks to see that the block is also read by a task somewhere in the system. If the block is not read, an error message is produced.

This restriction creates an extra burden for a designer who chooses to follow a top-down integration plan. In such a plan, the designer develops the software system as a series of progressive iterations where the first iteration provides ony the skeleton of the system and succeeding iterations add major functions one at a time. A result of such a plan is that there will be many COMPOOL blocks defined during any one iteration which will not be used until a later iteration is completed. Typically, a COMPOOL block that is generated (written) as a result of processing incorporated into iteration (n) will not be used (read) until iteration (n + 1) incorporates the set of tasks designed to use those inputs.

Normally these "stubbed" outputs present no problem to a designer, since they will all be picked up in later iterations. PALEFAC, however, makes no allowance for this possibility. The designer is unable to stub the outputs, so a dummy task must be created which does nothing but read the offending COMPOOL block. While this process is trivial for any single COMPOOL block, the complexities of a large system can create many extra hours of work for the designer.

It is recommended that PALEFAC produce only a warning message for this situation.

5.2.2.3.3 COMPOOL Block Update Affects Only Local Copy

When a task writes a COMPOOL block, the copy of the block residing in the local processor is updated. When PALEFAC processes each WRITE statement, it checks to see if a copy of the block being written also resides in a remote processor or terminal. If so, it generates the necessary bus message instructions to accomplish the transmission of the data to that remote copy of the block.

If a COMPOOL block is both read and written by a task, PALEFAC does not generate the bus message instructions to update any remote copies of the block. This causes the designer some problems. If the task in question is synchronous (scheduled to execute at a particular time on a cyclic basis) then the desired bus message can be specified by the designer to follow the execution of the task. But if the task is asynchronous, the designer must choose between one of two alternatives:

a. Specify the transmission of the block in the PGI file as being synchronous. This will result in the block being transmitted over the bus at regular intervals, even if the task in question which writes the block does not execute. This compromise increases the bus load unnecessarily.

b. Create another COMPOOL block ("B") which is simply a copy of the first block ("A"). When the task in question finishes updating block A, it will copy the data into block B. Then it will write block A and block B. This second write will cause PALEFAC to create the bus message instructions necessary to accomplish the transfer of the data to the remote processor or terminal. This compromise wastes core storage in the form of extra blocks and code and it wastes execution time.

If a task READs and WRITEs a COMPOOL block, PALEFAC should produce an asynchronous update message for all global copies of the block, not just the copy in the local processor. Also, the system designer should be notified by PALEFAC in the form of a warning message when such an update is generated. This will allow the designer to consider less costly alternatives for transmitting the data in question.

5.2.2.3.4 Asynchronous COMPOOL Blocks Can be Read in Only One Task

In the DAIS Mission Software Executive Specification (SA 20130?), figure 3.1.2.4.3-1 states that an asynchronous COMPOOL block which is only read in the processor, not written, may be read by only one task. This introduces some unnecessary design restrictions. If two tasks need this same input data, one task must pass this data to the other by copying it to a second block and then writing it. This wastes core storage and execution time.

It is recommended that PALEFAC only generate a warning message for this case. This will allow the system designer more flexibility while still helping him to maintain data integrity.

5.2.2.3.5 Synchronous COMPOOL Blocks Can Be Written In Only One Task

PALEFAC allows a synchronous COMPOOL block which is only written in the processor, not read, to be written by only one task (described in the DAIS Mission Software Executive Specification, SA 201302, figure 3.1.2.4.3-1). This is to prevent the possiblity of a second task attempting to update and write the same block before the synchronous bus message was executed to transfer the data generated by the first task.

The PALEFAC restriction thus ensures compliance with design standards that will guarantee data integrity for synchronous COMPOOL blocks. There are situations, however, where a slightly less rigid standard will still provide complete data integrity. One such situation is a program which has independent processing paths where only one path can be active at any time and where the same outputs are generated. As an example, consider a navigation program that has many sources of sensor data which can be used to generate a common output, such as an updated position vector. Each sensor source requires special tasks to handle its inputs and transform them into a standard format for processing. These tasks are not active unless the corresponding sensor has been selected as the data source. Thus, even though many tasks in the system are capable of updating the position vector, only one of these tasks can be active at any one time. Even though data integrity would be maintained in this situation, the restrictions in PALEFAC cause the designer to generate an extra task whose only function is to take each "different" position vector and generate a standard output.

It is recommended that PALEFAC only generate a warning message for this case. This will allow the designer more flexibility while still helping him to maintain data integrity.

5.2.2.3.6 LINKS Control Statements

The PALEFAC program is supposed to generate the control statements necessary to link-edit the program modules for the operational system. This feature works for the DEC-10 load modules but does not work for the HBC option of the linkage editor. It is recommended that this be implemented.

5.2.2.3.7 Bus Load and Predicted Executive Overhead

In its original concept, PALEFAC was to have provided a partitioning facility designed to allocate tasks to processors in an efficient manner. The result would have been a configuration with the lowest possible bus traffic and executive overhead. Because this is a complex task, it may not be practical to modify PALEFAC to perform it. There are however, two areas in which PALEFAC could be used to generate valuable input to a designer. The first is to calculate the bus load by minor cycle as called for in the Phase II development plan for PALEFAC. The second would be to predict executive overhead, both local and master, by minor cycle. These inputs would provide a designer with invaluable assistance in locating potential trouble spots in the execution sequence.

5.2.2.4 Software Test Stand Linker (LINKS) Problems

The LINKS program was effective in linking the files produced by the JOVIAL J73/I compiler and the HBC Processor/Cross Assembler. The run times associated with this program were reasonable considering the task that was being performed by this program.

The only problem associated with using LINKS was the diagnostic messages produced when errors were detected by the program. In most cases these error messages had little meaning to the engineers using the program and the messages did not appear in the documentation, "User's Manual for Software Test Stand Linker," MA 212200.

For example, the error message "Multiply Defined Global Detected" is produced when more than one global symbol of the same name is discovered, but the program does not indicate which of the symbols it will use to resolve references.

The program should be modified so that the diagnostic messages are more meaningful and provide some insight into debugging.

5.2.2.5 Performance Monitor and Control (PMC) Problems

The Performance Monitor and Control (PMC) program ran on the DARTS facility PDP 11/40 computer connected via the bus to the DAIS processors. The program capabilities include loading the processors from files stored on disk on the 11/40, modifying the contents of the DAIS processor core storage, and monitoring and recording selected bus transmissions for offline analysis.

The PMC program was never fully operational at the DARTS facility. Of its many capabilities, only the load and record features were used successfully. However, PMC is fully operational at AFAL.

Of the PMC features that were not operational at DARTS, two would have been especially useful. One is TALK, a feature which allows the PDP 11/40 to accept CIU commands at the 11/40 teletype and relay them to the CIU. Throughout the study, numerous patches had to be made to the software resident in the AN/AYK-15 processors. Every one of these patches had to be entered by hand at the CIU in a lengthy and error-prone process. (It should be noted that although the Hazeltine cassette unit was available to transmit patches to the CIU, it was unreliable and often failed in the middle of a load.) If the TALK feature had been available, a set of these patches could have been entered one time at the PDP 11/40 teletype and stored on disk as a command file. The patches would then have been available whenever needed simply by executing that command file in the TALK mode.

The second PMC feature that would have been helpful is UNLOAD. This feature allows a program residing in an AN/AYK-15 processor to be read and stored on a disk file of the PDP 11/40. This feature would have been used to save debugged and patched applications programs. Twice during the study software errors were located which could be corrected only by patching several hundred words of the program. It is not reasonable to enter this many patches into a program every time it is loaded. Since neither TALK nor UNLOAD were available, new load modules had to be generated by rerunning PALEFAC and LINKS. It is estimated that the lack of the TALK and UNLOAD features at DARTS cost two to three man-weeks in additional engineering effort.

5.2.2.6 DAIS Diagnostic Software (M$DIAGN) Problems

This software is executed on the DAIS processors in response to commands entered from the CIU keyboard. It is used as a debug aid to display various executive data. It can also read and display the DAIS BCIU registers.

The displays generated by this software were generally very helpful in diagnosing problems. The main problems encountered using this program were the errors in the displays (e.g., wrong values displayed on the P$TTB page) and the lack of documentation. The diagnostic software has the potential to be a most useful debugging tool. In addition to improving the documentation and correcting existing errors, it is recommended that the text accompanying the various displays be expanded to make them useful to all users of the DAIS executive software. As the displays now exist, a user must have considerable knowledge of the executive design in order to make full use of the information presented, but as the executive evolves, the users' necessary level of knowledge about its internal design will decrease, so expanded diagnostic capability will be of great importance to the users' debugging efforts.

5.2.2.7 ALAP Problems

The version of ALAP (Hot Bench Computer Assembler Program) used during this study was the same one being used at the Avionics Laboratory at WPAFB, Ohio. The program performed the function of reading code written in an ALAP specified assembler language and produced executable machine language programs for the AN/AYK-15 processors. The "ALAP specified" assembler language should be stressed since ALAP is actually a cross assembler and one of the inputs to the program is a specification of the assembler language to be input to the program.

5.2.2.7.1 ORGIN Statement Error

The ORGIN statement did not work as intended. Many attempts to use this particular instruction to produce listings relative to realtime memory residency were not successful. ORGIN values less than 1000(16) and larger than 8000(16) failed. The diagnostic message produced by the program was not meaningful since it only reported that the value was too large; however, 5000 and 7F00 both were used successfully but 1000 was "too large" as was 40 . The ORGIN statement is of great value, so this problem should be examined and corrected in future versions of ALAP.

5.2.2.7.2 Diagnostic Messages in Interactive Mode

When operating ALAP in interactive mode (via a remote terminal), the program would find errors in the source code and report that errors had occurred; however, the source code line number(s) in error was not provided. So the programmer had to wait until the ALAP output file was printed (up to 12 hours) or display the file on the terminal itself in order to determine what the error was. These two alternatives both proved costly.

The program should be modified to produce an error file which contains only selected error messages and the source statement sequence number. This file should be made available to the interactive terminal user for realtime problem determination and correction.

5.2.2.7.3 Abnormal Program Halts

On three occasions during this study the ALAP program simply halted and displayed the DEC-10 program counter. No more information was provided and attempts to rerun the program produced the same result. In all three instances the resolution was to re-create the input file in its entirety and rerun the program.

It is recommended that more run-time information be provided at the user's terminal.

5.2.3 Problems Encountered With DAIS Support Software Documentation

5.2.3.1 J73/I Compiler Documentation Problems

Our compiler documentation was the JOVIAL J73/I Computer Programming Manual (October 1977) which adequately described the syntax of the language features. The lack of applicable examples, however, added to our software development time significantly. A detailed syntax chart is of limited value without adequate examples of usage.

Also, the error messages issued by the compiler need to be better described in the manual. What the programmer needs, particularly the novice, is a real description of the error related to the obviously violated syntax rule. This is especially true in the area of data structure definition.

Additionally, we encountered several restrictions which were not documented in the manual. These may be errors in the compiler itself, or omissions in the documentation. The following paragraphs describe these problems.

5.2.3.1.1 IN Attribute With GLOBAL Copies

Any program which has the IN attibute on the PROC statement (as required by the DAIS Software Development Standards) will produce a fatal and ambiguous compiler error message if it also references a COMPOOL block using the GLOBAL'COPY directive. This is not documented.

5.2.3.1.2 Table Items in OVERLAY Statements

The compiler does not allow table items to be used in OVERLAY statements. This is not documented.

5.2.3.1.3 IN Attribute With Local Tables

The compiler produced ambiguous error messages when the IN attribute was used with local tables. In one case the compiler allowed the IN attribute only on the first-declared local table, giving an "illegal allocation specifier" error message for a subsequent local table declaration with the IN attribute. The compiler then gave a "missing semi-colon" message for the next declaration.

5.2.3.1.4 RESERVE Data

Any program data which is RESERVE by default may not be explicitly declared RESERVE, since this causes ambiguous and undocumented error messages.

5.2.3.2 PALEFAC Preprocessor Documentation Problems

The documentation for this program consisted of the Interface Control Document (ICD), PALEFAC Preprocessor/PALEFAC to Mission Software, SA 802309C, and the User's Manual for PALEFAC, MA 202200B.

Together these documents proved generally satisfactory for using the preprocessor effectively. Specific problems encountered which should be corrected, are detailed in the following paragraphs.

5.2.3.2.1 SCHEDULE Statement Inputs

The Interface Control Document (ICD, PALEFAC Preprocessor/PALEFAC to Mission Software, SA 802309C, paragraph 3.1.3.2.1.5, p. 60) states that the task parameters for a SCHEDULE statement may appear in any order. This is incorrect. For example, in a privileged mode task, the PRI=PRIV field must appear before an UPON . In a normal mode task, however, the PRI parameter may appear anywhere. The documentation also states that the priority field is mandatory, but the preprocessor does not check to see if the field is present.

5.2.3.2.2 Two SCHEDULE Statements for the Same Task

The ICD (paragraph 3.1.3.2.1.5, p. 60) states that if two or more SCHEDULE statements are written for the same task, only the first statement is examined by the preprocessor. This is incorrect. The preprocessor examines every SCHEDULE statement present in the code. If the same task is referenced in two of them, the scheduling information provided by the second statement replaces the data from the first. Thus, if there are several SCHEDULE statements for a single task, the scheduling data passed on to PALEFAC is that contained in the last statement examined by the preprocessor. This behavior was documented in preprocessor runs made during the course of the study. The ICD should be corrected to reflect the behavior of the preprocessor.

5.2.3.2.3 PMI Record Format

The PALEFAC Preprocessor Detailed Design Specification (PALEFAC PP DDS, SA 202201, figure 3.2.3.b, p. 6) incorrectly describes the content of byte 8 in the PALEFAC Module Input (PMI) file record. The documentation should indicate that the field contains a - if the scheduled task priority is relative, a 0 if it is absolute, and a b if it is privileged.

The description for bytes 9-12 incorrectly shows the format as bbb0 if the task is privileged. The correct format is bbbb.

5.2.3.2.4 Subfunction Control Flow

The PALEFAC PP DDS describes a number of subfunctions on page 14. It is unclear from the documentation how these subfunctions relate to one another. It would be helpful to the reader if the control function flowchart on page 12 were updated to show how the subfunctions fit into the overall scheme.

5.2.3.2.5 Trim Subfunction

The flowchart describing the trim subfunction (PALEFAC PP DDS, SA 202201, figure 3.3.2.1.4.a, p. 18) contains a number of errors. These errors, which appear both in the equations and in the flow, make it difficult to use the flowchart. Without the flowchart, the reader is unable to determine how the trim subfunction affects the JOVIAL source code. This information is important because the trim subfunction may be at least partly responsible for the very lengthy processing times required for preprocessor runs.

5.2.3.3 PALEFAC Documentation Problems

The PALEFAC program documentation was the Interface Control Document (ICD), PALEFAC Preprocessor/PALEFAC to Mission Software, SA 802309C, and the User's Manual for PALEFAC, MA 202200B.

These two documents were generally satisfactory for using the PALEFAC program effectively . Specific problems encountered, which should be corrected, are detailed in the following paragraphs.

5.2.3.3.1 Module Naming

Each applications module possesses two names. One is the four character "common" name used for cross-referencing by other modules, the second is the multicharacter extended name which is used to identify a particular version of the module (if more than one exists). These two names are not interchangeable in their use, a circumstance which can lead to some confusion.

Although the User's Manual for PALEFAC (MA 202200B) recognizes this problem by supplying an appendix C on Naming Application Modules, the appendix is incomplete and the body of the User's Manual makes inadequate reference to it. For example, the PALEFAC Global Input (PGI) file specifies which modules are to reside in each processor. But the PGI writeup (paragraph 3.1.3, p. 12) neither tells the user which name should be used nor refers to appendix C for the information. In contrast, the description of the PALEFAC Auxiliary File (PAF) does describe which name is to be used, but again, no reference is made to appendix C. In this case, the omission is probably wise since appendix C does not mention which name is to be used in the PAF. One other clarification should be made to appendix C. Unlike other JOVIAL names, the extended task name may not contain an apostrophe. The PALEFAC preprocessor treats the apostrophe as a delimiter and truncates the name at that point. This preprocessor behavior was discovered during one of the first runs conducted for this study.

5.2.3.3.2 PGI Bus Messages

The bus message segment of the PGI (User's Manual for PALEFAC MA 202200B, paragraph 3.1.2.4, p. 20) contains an activity register (ACTREG) field. The description provided for this field is inadequate. Without supplementary information, a user is unable to determine the nature of the data to be supplied.

5.2.3.3.3 PAF Format

The User's manual is inconsistent in specifying the format required for the blocksize input to the PAF. The PAF description (paragraph 3.1.3, p. 23) states that the field is in column 9. This is ambiguous since the field can contain two numbers. Does the field start or stop in column 9 and is it right- or left-justified? The example (A.7, p. A13) provides no help whatever since it shows the blocksize field starting in column 14. To further confuse matters, the PAF file used during this study executed successfully when the first character of the blocksize field was placed in column 11. To assist the user in creating the PALEFAC input files, each field should be described in terms of its maximum extent, and whether data contained within that field must be right- or left-justified.

5.2.3.4 Software Test Stand Linker (LINKS) Documentation Problems

The documentation for the LINKS program was the User's Manual for Software Test Stand Linker, MA 212200. This is a seven page manual with a twelve page appendix - extememly poor documentation for such a key program. Almost nothing is explained adequately, and no error messages are documented in any way. This lack of documentation cost us many man-hours during the first few of our system builds.

5.2.3.5 Performance Monitor and Control (PMC) Documentation Problems

The Performance Monitor and Control (PMC) Executive User's Guide (MA 207301), does not provide sufficient detail to allow an inexperienced person to use the system without outside assistance. In addition, there are errors in documentation which could cause difficulty for a reader. Specific problems, which should be corrected, are detailed in the following paragraphs.

5.2.3.5.1 Implied DEC-10 Availability

An implicit assumption is made that a DEC-10 computer is linked to PMC. This assumption does not apply to users lacking an in-house DEC-10 facility.

5.2.3.5.2 Lack of Hardware Configuration Information

No information is provided on the hardware configuration which PMC supports.

5.2.3.5.3 Incorrect CIU Command Format

The example given for the CIU command is CIU $2, whereas the format used during the study was CIU 2.

5.2.3.5.4 No Default Information

No information is provided on the defaults supplied in PMC. For instance, the CIU command need not be used if CIU 1 is desired after loading PMC. The CONFIGURE command is also presupposed by the defaults, as well as a multiprocessor configuration. In short, a person who runs PMC to load a Westinghouse processor through CIU 1 must, according to the User's Guide, sequence through the following commands:

```
CONFIGURE
Westinghouse Processor #2
SCADUS?
URT? (Answer with Y or N)
SSDF?
MODELS?
DEC-10 TALK?
BMU?
4-Port Buffer Memory?
CIU 1
LOAD XXXX.XXX
```

In fact, the user need only enter the LOAD XXXX.XXX command to achieve the desired result.

5.2.3.5.5 No RT-11 Information

No information is provided on the RT-11 under which PMC operates, nor is there any reference to RT-11 manuals which may be available.

5.2.3.6 DAIS Diagnostic Software (M$DIAGN) Documentation Problems

Documentation was not provided for this program. This software package was a valuable debugging tool, but much better use could be made of it if it were carefully and fully documented.

5.2.3.7 ALAP Documentation Problems

The document provided was DAIS Processor/Cross Assembler User's Manual, MA 206200, 22 December 1976. The documentation for the ALAP program proved to be generally weak. For instance, to use the program as Boeing used it (to produce object code for an AN/AYK-15), the legal instructions or operation codes are found in the "DAIS Processor Instruction Set," SA 401301; however no mention of or reference to this manual appeared in the documentation. Specific problems, which should be corrected, are detailed in the following paragraphs.

5.2.3.7.1 ALAP Directives

In section 7 of the User's Manual a number of directives for the program are listed, however nowhere in the document are these directives discussed. No explanation or definition of operands is provided.

5.2.3.7.2 Syntax Rules

Section 3.0 provides a set of syntax rules for each input statement. This set of rules provides the format of the input statement but very little about the rules for the formation of the fields in this format. For example, the discussion of the "ARGUMENT FIELD" contains references to "one or more subfields;" however, no discussion or description of these subfields is provided.

5.2.3.7.3 Program Operation

The instructions provided for operating the program (such as input or output files, program switches, or options) will not work. The format of the RUN command successfully used to run the program does not agree with the format given in the example.

5.2.3.7.4 Missing Section

There is no section 6 in the document. Section 7 is labeled "section 6" and is preceded by section 5.

5.2.3.7.5 Erroneous Error Messages

Appendix A contains an error list; however some of the error messages are not clear. In fact, some of the error messages refer to directives not even defined for the version of ALAP used in the study. Examples include:

EXTRNS (is this EXTERNAL?)
ESTORAGE (?)
ORG (is this ORGIN?)

These examples are from error message 11. Does this error message really mean that external references are not allowed? For interactive users, these messages should reference an input line number.

5.2.3.8 DAIS Processor Instruction Set Documentation Problems

The documentation provided for the DAIS processors was the DAIS Processor Instruction Set, SA 401301. This manual has several key omissions, needing correction, which are detailed in the following paragraphs.

5.2.3.8.1 Execute Instruction

The description of the execute (EX) instruction (DAIS Processor Instruction Set, SA 401301, p.51) does not mention that the memory address must be an even location. If a user attempts to use an odd memory address, the processor will halt in a manner that does not allow the user to locate the instruction in error. The EX description should be amended to include the even memory address requirement.

5.2.3.8.2 Console I/O Instruction

The description of the console I/O to peripherals (CIO) instruction (p. 53) does not include timing. Since the processor waits until the console operation is completed, the timing could become important to a designer. As a case in point, the master executive uses the CIO instruction to generate messages for display on the Hazeltine consoles. During this study, one frequently seen message was "M$BINT: XSWR = 1 IGNORED." This message was often associated with irregularities in program operation. What was not known was how the act of displaying the message was linked to those irregularities. When the program irregularities proved difficult to trace, the display of the message was eliminated from the executive code and several of the previously unresolved problems immediately disappeared.

It became apparent that at least some of the problems were a result of delaying program execution in order to produce a message display at the console. Since no timing estimates were available for the CIO instruction, it was impossible to anticipate this problem. Until timing estimates are made available to a designer, the only safe procedure to follow during program development and test is to eliminate all such displays from the DAIS executive.

5.2.3.8.3 Floating Point Instruction Timing

In the "DAIS Processor Instruction Set" (SA 401301), instruction timings are provided which are useful to the designer for estimating timing requirements. The instruction timings for floating point arithmetic, however, are incomplete as each is expressed as a constant plus "function of numbers." "Function of numbers" is not defined and since floating point arithmetic can be time-consuming the designer is left without critical timing information.

5.2.3.8.4 Input and Output Extended Mnemonics

A set of extended mnemonics are provided for the input and output instructions such as DSBL and ITB. These are not defined in the Instruction Set Manual.

6.0 Recommended Changes and Modification Considerations

This section discusses recommendations for possible changes or modifications to the DAIS Executive Computer Program that could be considered for future implementations of the program. The majority of these recommendations are based upon analyses and trade studies of military aircraft requirements. They were formulated using the 1977 version of the DAIS Executive Computer Program.

AFAL has been and is continually developing this program for future use on aircraft within the Air Force inventory. Consequently the program is constantly being upgraded in the laboratory and as a result a number of the changes recommended here are already being implemented or are being considered for implementation under separate efforts.

Paragraph 6.1 discusses change considerations relative to the version of the program which was provided to the contractor in September of 1977 for evaluation. Paragraph 6.2 discusses change considerations for advanced versions of the program for support of avionics systems in the future.

6.1 1977 Version of DAIS Executive Computer Program

The 1977 configuration of the DAIS executive was determined to be efficient, considering the numerous tasks it performs.

A shortcoming of the program as evaluated was in the area of error handling and recovery (EHAR). EHAR, as implemented, consisted almost entirely of recognizing an error, reporting the error, and halting the program. EHAR for an operational flight program (OFP) requires graceful degradation and reconfiguration features to enable the most critical functions to continue operating as long as possible. Reliability is one of the most crucial features in aircraft systems and necessarily must be included in the airborne software system design.*

During application software design with the 1977 executive program, a software system was built around an avionics system architecture to operate under control of a standard reusable executive computer program. This was accomplished through a very rigid interface between the executive and the applications program. This interface is enforced and maintained through a set of static tables which reflect an optimum state of the operational system. Being static, the tables cannot be reconstructed in realtime to handle the dynamic needs of various failure modes and reconfiguration requirements.

The rigid interface is one of the key aspects of the DAIS executive and adds to the usefulness of the program. This rigid interface makes it possible to design modular, reusable software applications programs, however, this rigidly defined interface did not necessarily have to be carried through into the coded configuration to be effective. If the format and syntax for a realtime interface between the executive and application software can be defined and maintained, for example, then the applications software can be moved from one application to another providing the same interface exists in the new application. The 1977 implementation goes far beyond this in that real time parameters are, in

*EHAR provisions are included in the 1979 version of the DAIS Executive.

effect, hard coded addresses. This particular implementaion certainly maintains the interface, but it excludes any dynamic or realtime control of the operational system through a data base which reflects the realtime state of the system.

6.1.1 DAIS Local Executive

6.1.1.1 Static Task Tables

Probably the major problem associated with using the DAIS local executive was the ordering of executable tasks into a static sequence so that they would execute as required. This problem was not so apparent when executing a fixed set of tasks such as those associated with an enroute navigation function; however, if some other function was required to augment the navigation function and had to operate interleaved in the sequence, the function then had to be placed in the static sequence at the time the system was constructed. This is due to the static characteristic of Task Table B and the fact that the order of Task Table B dictates the execution sequence of tasks.

It was extremely difficult ot integrate into the system applications tasks which did not operate in a truly cyclic manner, but operated when required, by the completion of other tasks. For example, it was preferable, from a performance point-of-view, to compute a computed air release point (CARP) when navigation data became available, not at the end of a cyclic navigation function processing. Another difficult problem to solve was the integration, into the operational sequence, of the processing of sensor data which was provided at a lesser update rate than other sensor data. A good example is Global Positioning System (GPS) data which was provided on a six second basis. When the GPS data was available (every six seconds) the software programs which operated on this data needed to be interleaved with the free inertial navigation tasks. In order for these tasks to operate in the proper sequence, a fairly difficult set of signals and activation events had to be devised. This could have been implemented by including the GPS processing algorithms in two free inertial navigation tasks. This is not an unusual solution, but it is not in keeping with the principle of modular reusable software since the software cannot now be readily moved to a different avionic application without GPS.

Modification of the executive to handle dynamic scheduling requirements rather than servicing static requests which are frozen during design should be considered. If this executive is to be used in a tactical weapons system, the task for a designer to compute and account for, in a static table structure, all the combinations of computational requirements will be difficult.

A possible implementation includes handling static cyclic dispatch lists which are activated or presented to the executive by a knowledgeable controller and demand or noncyclic tasks are scheduled dynamically and then executed immediately.

6.1.1.2 Event Service

Another problem closely allied with the static task table problem is the complexity of the event service which is forced by a static execution sequence.

For example, suppose tasks A and B are routines scheduled by controller C1 to run in sequence A-B. Also suppose that under certain conditions another task AA is to execute immediately after task A (perhaps to modify data written by A and read by B). Due to structured design considerations, task AA is scheduled by a different controller, C2, which has a higher priority than C1. Thus, task AA has a higher priority than tasks A and B.
In order to establish the execution sequence A-AA-B, the designer might consider the following alternatives:

1) Put the completion of task A in the condition set for activating task AA. Unfortunately this is not supported by the DAIS executive as evaluated.

2) In task A, issue a SIGNAL of an unlatched event which is in the condition set of task AA; but because AA needs data written by A, this SIGNAL would have to come after the WRITE statements in A, which would violate the Software Development Standards as reported in section 5.0. This alternative also presents a problem if task AA is scheduled with the same minor cycle event as A in its condition set. In this case, task AA would never execute since the minor cycle event must be the last event satisfied in a condition set in order for the task to be made active.

3) In task AA, as the first executable statement, issue a WAIT statement for the completion of task A. This is also a violation of the standards and results in more tasks waiting than would seem desirable, but it does represent a solution.

This example serves to illustrate the complexity of condition set and event interplay. This complexity could be reduced by the use of dynamic scheduling, in which a task would become eligible for execution immediately upon being scheduled. The current DAIS execution sequencing design should be reexamined and possibly replaced with a dynamic scheduling capability. Event service as it currently exists could probably be simplified to a WAIT service.

6.1.1.3 Lack of Local Input/Output Capability

As the DAIS executive is designed and implemented with the static tables (task table B, event tables, minor cycle event tables, etc.) the program is quite large primarily due to the size of the static tables. For example, the DINS program including the local executive and the necessary interface tables in the remote processor was in excess of 45K, which forced the use of a 64K processor. In the same application the memory requirement in the master processor for the master executive was 6.5K, the local executive 12.3K, and the interface tables 2.5K, for a total of over 21K. With memory requirements such as these, program overlays and realtime storage devices are definitely necessary to support the applications programs.

In the present configuration and design the program has no capability to support an online device. The current DAIS processors do not have this capability either. Yet this capability is desirable in many airborne applications.

To support overlay programs stored on an online disk or drum and possibly to support realtime mission data recording a capability to handle input and/or output from devices such as disk, drum or magnetic tape is necessary.

6.1.2 DAIS Master Executive (Bus Controller)

6.1.2.1 Error Handling and Recovering (EHAR)

If a subsystem or an RT were to fail, there is no intelligence provided to the system by the master executive aside from a table local to the master executive being updated and bus instructions for that device being NO-OP'ed. The master executive has no information whatever relative to the tasks which use the RT data.

A method of communicating with both the local executive and with the application program dependent upon data from a specific device, needs to be implemented to ensure proper system operation. The EHAR issue needs to be examined in more detail and a means provided to support requirements for avionics applications.

6.1.2.2 Lack of Test Capability

When operating a program in an integration test or debug mode, there is no real method of halting the program and resuming from the point of interrupt. In those cases where inspection of realtime data is necessary, an attempt to halt a remote processor for data inspection results in the remote processor being failed and the bus instructions associated with that processor being NO-OP'ed. Resumption of program operation requires either a reload of the master processor or very tedious patching to return the data base to the original state.

One possible method of solving this problem would be to utilize a debug switch which would be checked by the master executive in the event that a remote processor did not respond in the alotted time. At the time of recognition, the master could send an idle message to the other processors (which would not contain any minor cycle updates) and then poll the nonresponsive processor. At the time of initiating the poll of the nonresponsive processor, a display could be generated to the Console Intelligence Unit (CIU) to inform the operator. If the "failed" processor did respond, the master executive could then send a special interprocessor service request which would inform the other remote processors of the resumption and a minor cycle update to resynchronize the system. If, in debug mode, the processor did fail, rather than simply halt, the system would probably have to be reconfigured anyway.

Another possible method of implementing this action would be to effect a failure in debug mode through a time delay or an operator action to allow "real" failure testing.

6.1.2.3 No Reconfiguration Capability

Reconfiguration has been implemented in the EHAR design as a method of reloading a different system configuration from a mass storage device on the MUX.

Failure processing (in the 1977 version) consists of reporting the error and dropping the miscreant terminal from the configuration. One of the major obstacles to attaining a reconfiguration capability is the fact that the system configuration is frozen in the design stage by a support software program which precedes operational software production by at least two steps.

This software support program is the PALEFAC program which produces tables that provide and maintain the interface between the DAIS Executive Computer Program and the applications program. These tables are rigidly constructed and do not allow reconfiguration because of this rigidity. A system is built (by PALEFAC) on the assumption that a particular program will reside in a given processor and the source and/or destination of a message will always (for the life of an operational system) be the same. The PALEFAC program does not "build" an operational system but constructs an executive data base to represent a single configuration which is static and cannot be modified in realtime. In order to build another operational system, the PALEFAC program is run a second time using a different data base which reflects another single configuration.

The DAIS executive and applications software interface should be reexamined with the goal of making it less rigid for the purpose of reconfiguration. Attempts have been made to solve the reconfiguration problem by using different load modules for each of the various possible system configurations, all residing on a DAIS mass storage device. This is a possible solution to the problem, but in actual practice this method will probably be severely limited by available space on the mass storage device. This is because a different software system must be built for each hardware/software configuration to be supported. This includes the PALEFAC processing, LINKage EDITing, compilation of interface tables and the loading of all of the different software systems onto the mass storage device.

Another method which should be examined is a dynamic reconfiguration method in which an initialization (or reinitialization) program determines the hardware configuration and then customizes the software to fit. Algorithms similar to those used by the PALEFAC program could be used. The problem with this approach is that almost all realtime executive requests are translated into static offsets in the static tables generated by PALEFAC. If reconfiguration were to be implemented with this approach, some dynamic executive service request scheme would have to be employed. Methodology such as this has been successfully implemented in a number of airborne systems in the past including E3A and E4B.

6.1.2.4 Realtime Handling of Asynchronous Transmission Requests

At the present time, the DAIS master executive (bus control) program upon recognizing a status exception (request for asynchronous activity) will suspend the synchronous activity at the end of the current transmission. This is a costly and wasteful implementation that introduces an unnecessary degree of complexity to the program. The need for a nonsynchronous transmission is perhaps necessary, but the present

implementation interrupts the synchronous bus list processing and inserts a new message sequence into the bus list (I/O instructions). This asynchronous processing could probably be handled as easily at the end of the synchronous bus list processing. By doing the asynchronous processing at this point, a degree of complexity is eliminated from the program. The time in a minor cycle after processing the synchronous bus list is free time, since nothing but idle polling is taking place. An implementation of this sort would delay the response to these "asynchronous" requests by a maximum average of 0.5 minor cycles. Idle polling is a query for any asynchronous message transfer requests.

Interruption of synchronous bus list processing to handle an asynchronous message seems to be inconsistent with the system design philosophy, which is based on cyclic or synchronous processing.

A situation to cite as an example occurred during PINS testing where a number of asynchronous requests were pending and the resultant processing skewed system synchronization to the point of failure. Although PINS was purposely designed to introduce heavy loads to the executive, even a well-partitioned operational flight program could experience heavy asynchronous loads during transitory periods (such as those induced by mode changes) when a number of signals must be set, data blocks written, and tasks cancelled and scheduled.

6.2 Change Recommendations for Future Programs

The DAIS Executive Computer Program in the present configuration will support a limited number of avionic architectures. The executive is limited to supporting those architectures that have the following limiting features:

- AN/AYK-15 or Magic 362F Processors
- 1553A/DAIS Bus Protocol
- Federated, single level bus architectures
- DAIS BCIU and RT interface hardware

The trend in avionic architecture arrangements are those using multilevel buses and microcomputers that use microprocessors. Also, the MIL-STD-1553A/DAIS protocol has already changed to MIL-STD-1553B. Obviously, the limiting features listed above are not compatible with future avionic architecture requirements. Changes or modifications will have to be made to the program to accommodate multilevel bus, microcomputer-based distributed architectures.

To assist in upgrading the DAIS Executive Computer program to accommodate the systems of the future and to be consistent with the DAIS philosophy, the following change or modification recommendations are presented:

- Machine Independence
- Architecture Independence
- Modularity

6.2.1 Machine Independence

It is apparent that machine independence was considered when the program was developed. For example, the majority of the program was coded in higher order language (HOL). In addition to the program being implemented in a theoretically retargetable language, the processor hardware interfaces are fairly well isolated in code, however, the processor dependent code is not isolated well enough to make the rest of the executive machine independent.

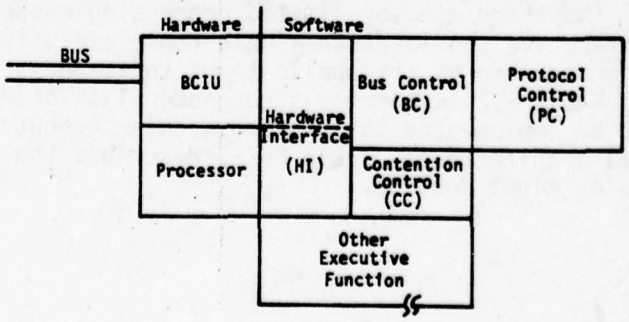
The proposed MIL-STD-1750 attempts to provide machine independence. If it does become a standard, some of the interface problems will be minimized, and perhaps relative to the target computer the program will become machine independent, however the hardware dependency is not entirely in the interface to the computer itself. Probably a more confining or restricting interface is with the BCIU. The DAIS BCIU operates off a set of programmable registers which must be set in software. In some cases tables must be generated in software to satisfy this interface.

It is recommended that the hardware interfaces (e.g., interrupt handlers and timers) be totally isolated into program modules. There should be one module for each piece of hardware and the interface with the remainder of the executive should be rigidly defined and rigidly enforced. Using this methodology, if a piece of hardware is changed for an application then only the module which deals with that piece of hardware need be changed.

6.2.2 System Architecture Independence

The present DAIS executive was designed for only one system architecture, which is a single point control federated system operating with 1553A/DAIS protocol on a single level MIL-STD-1553A MUX bus. If the executive program is to be reusable (in keeping with the DAIS philosophy), then the program should be modified so that it can easily be implemented with other protocols or system architectures. To achieve this goal, the DAIS executive must be modularized to be a family of independent functions so that selected modules can support a variety of system architectures. Possible modularity is discussed in 6.2.3.

Bus control is basically a set of functions which include the hardware interface and the software response to the protocol. In some cases, the bus control may also include some sort of contention control, but whether operating in a system with single point control or contention, once BCIU control is established it is independent of protocol. Graphically this appears as:



If the protocol were to change from 1553A/DAIS to 1553B, only module PC would have to be replaced. If a contention system was desired, then the appropriate CC module would be included. The interfaces are rigidly maintained so that the individual modules can be built to be independent of the rest of the executive.

6.2.3 Modularity

In addition to attaining independence from hardware or system architecture, modularity can simplify maintenance; each function is built into a functional building block with very rigid interfaces, which isolates the functional area requiring maintenance from other functional areas.

There is a practical degree of functional modularity that can be attained for an executive program. First program complexity and size must be considered. For example, if the DAIS executive in the maximum configuration was to be treated as a single reusable program and used in a microprocessor-based intelligent sensor, it would probably force the memory size of the microprocessor to 32K and perhaps double or triple the necessary processing capability. The intelligent sensor would then contain elaborate task control algorithms and interprocessor service request algorithms. Probably the microprocessor-based intelligent sensor would need, at most, limited task control, bus interface control and local I/O handling software functions to augment its processing capability. As the DAIS executive is implemented, two choices are available. One, develop new executive programs to handle these functions (which is not in keeping with modular reusable software) or two, purchase a larger microprocessor to house the unnecessary executive functions. An alternative solution would be to modularize the DAIS executive so only the necessary functions need be included in the operational executive for an application. The degree of modularity for a program of this type would necessarily have to be traded against the implicit added overhead due to modularity (e.g., additional interfaces such as CALLs and linkages).

Executive control requirements for distributed multilevel hierarchical architecture differ little from an exclusively federated single level architecture except in three areas: (1) multiple buses must be controlled by a master bus controller, (2) variations in bus control philosophy may be used to control the bus depending on the application and (3) processors of limited capability will be used in distributed architecture. Each of these areas requires more modularity and isolation between modules in the software than the DAIS/executive currently contains. Areas (1) and (2) imply that the bus control software has a well defined interface which separates the bus control from the local executive. The third area of limited processing capability implies that the local executive should contain only those executive support capabilities required by the applications software in that one processor. Table 6.2-1 contains a suggested list of executive modules which could be implemented to create a modular executive for distributed hierarchical architectures Table 6.2-2 describes the functions that each of the modules might include.

PROPOSED EXECUTIVE MODULE	CURRENT RESIDENCE	REMARKS
Processor Interface	Master Executive	May never be standard but should be isolated
BCIU Interface	Master Executive	
Bus Control 1553A/DAIS	Master Executive	Assumes that control is established if necessary
Bus Control 1553B	n/a	
Contention Control - Polling/Handover - Poling Response/Takeover - Contention/Takeover - Contention/Handover - Time Slice/Takeover - Time Slice/Handover	n/a	Should all be separate modules and included only if needed (i.e., single point none are needed
Task Control	Local Executive	All functions now included whether needed or not
Bus Interface/Response Data Control	Local Executive	
Time/Event Control	Local Executive	
Interprocessor Service Request	Local Executive	
Asynchronous Transmit	Local Executive	
Asynchronous Receive	Local Executive	
Local I/O (Disk/tape)	n/a	
Local EHAR	Local Executive	Almost nonexistant
Global EHAR	Master Executive	Immature reconfiguration limited

Table 6.2-1 Suggested Executive Modularity

MODULE	DESCRIPTION
Processor Interface	Includes interrupt handling and communication with the processor for interrupt vector initialization.
BCIU Interface	Handles the setting of BCIU control registers and responds to BCIU interrupts.
Bus Control (1553A/DAIS)	Handles bus control protocol as specified for Mil-Std-1553A/DAIS. This includes response to and handling of all DAIS mode codes.
Bus Control (1553B)	Handles bus control protocol as specified for Mil-Std-1553B. This would include response to and handling of all mode codes.
Bus Control (Contention)	
- Polling Handover	Would poll all potential bus controllers after controlling the bus for a set of message sequences and relinquish control in response to a positive poll response
- Polling Takeover	Would handle positive response to a poll for control and would establish bus control and handle bus control for a set of message sequences.
- Contention Takeover	Would handle demand takeover of the bus control as it became available.
- Contention Handover	Would handle the relinquishing of control after a message sequence.
- Time Slice Takeover	Would handle the establishment of bus control based on a time interval (Round Robin).
- Time Slice Handover	Would handle the giving up of control after a specified time interval.
Task Control	Handles the scheduling and dispatching of tasks local to a processor. Included would be the executive service request handlers.
Bus Interface/Response Data Control	Handles data control local to a processor and respond to the bus controller (Master).
Time/Event Control	Will handle WAITs, timing and event signal and control local to a processor.
Interprocessor Service Service Request (IPSR) Handler	Will handle interprocessor service requests. These are requests for executive service which are satisfied in another processor. This module would handle both the transmission and the receipt of IPSRs.

Table 6.2-2 Executive Module Descriptions (Page 1 of 2)

MODULE	DESCRIPTION
Asynchronous Transmit	Handles asynchronous message transmission.
Asynchronous Receive	Handles asynchronous message reception.
Local I/O	Handles local I/O devices including initialization and interrupt response.
Local EHAR	Handle errors local to a processor.
Global EHAR	Handles system or global errors in conjunction with bus control and would effect reconfiguration.

Table 6.2-2 Executive Module Descriptions (Page 2 of 2)

Appendix A

Test Control Tables

The Test Control Table (TCT) controls the processing load that the Pseudo-Integrated Navigation system (PINS) places on the DAIS executive through executive service requests. Section A.1 of this appendix describes the features of the TCT and how it can be used. Section A.2 describes in detail the four Test Control Tables used in Phase 1 of this study.

A.1 Introduction to the Test Control Table

The TCT provides the test controller interface to PINS. Through the TCT, the test controller can control the individual components of the processing load requested by the PINS program. By entering the appropriate set of values in the TCT, the user may control both the absolute number of realtime statements resulting in executive service requests and the minor cycle in which these statements will be executed.

Each entry in the TCT defines the control for a test phase (a test phase lasts one second). Each test phase is independent of any other. For example, READs and WRITEs may be examined in one test phase, SCHEDULEs and CANCELs in a second, and SIGNALs in a third, or all three may be examined in a single test phase. The TCT can provide the most effective insight into the operation of the DAIS executive if phase control definition causes the executive load to vary in a gradual manner. Thus one phase might generate a single READ in minor cycle 2, while the next phase generates two READs in minor cycle 2 and a third phase generates three READs in minor cycle 2; or one phase may be designed to study one WRITE in minor cycle 4, while a second phase studies the WRITE combined with a SCHEDULE, and a third phase combines the WRITE and a SIGNAL. By carefully controlling the processing load in this manner, the user may obtain detailed information on the performance characteristics of the DAIS executive.

A complete discussion of the structure of the TCT and the manner in which the TCT controls the PINS processing load is provided in paragraph 3.3 of "Executive Evaluation Software Design", CRDL #1. Also found in paragraph 3.3 are definitions of special terms (e.g., base load) that are used in the descriptions that follow.

A.2 Detailed Descriptions

During the study, four TCTs were used to generate the data found in this report. These four tables have one common characteristic - each begins with a calibration phase consisting of what is referred to as the "base load". Phase 1 of each TCT is the calibration phase. A description of the calibration phase (base load) is presented in figure 3.1-5.

A.2.1 Test Control Table SACIA

Test Control Table SACIA was designed to study the effect of SCHEDULE and CANCEL statements when they are encountered in a variety of processing load levels. Figure A.2.1-1 shows the background load (0, 1, or2) that is executing during each test phase, the number of extra "events" occurring in each test phase, and the minor cycles (modulo 8) in which these extra "events" are to occur. As used here, "Event" refers to a request for an executive service, such as a READ or a WRITE.

Phase 1 is the calibration phase and phase 2 permits the study of a single SCHEDULE and CANCEL in a minimally loaded minor cycle (minor cycle 7). The remaining phases examine how the SCHEDULE and CANCEL statements perform in the three different background load levels and in two different local processing (local write) load levels.

Figure A.2.1-2 provides an explicit listing of the number of executive functions being performed in each minor cycle of each test phase of TCT SACIA. In each block of figure A.2.1-2, the test phases are arranged in the following pattern:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Each value in a block of figure A.2.1-2 represents the number of occurences of the corresponding parameter in each minor cycle for the test phase indicated by the value's pattern position. For example, in minor cycle 1, of test phase 16, 4 READs, 1 local WRITE, 1 global WRITE, 0 SCHEDULEs and 0 CANCELs will occur.

An empty block in figure A.2.1-2 indicates that the particular "event" was never executed in the corresponding minor cycle.

PHASE	BACKGROUND LOAD	LOCAL WRITE	MINOR CYCLES	
			TRIGGER	SCHEDULE AND CANCEL
1	0	0		
2	0	0		7
3	0	0		0,2
4	0	1		0,2
5	0	0		0,6
6	0	1		0,6
7	1	0	6	0,1
8	1	1	6	0,1
9	1	0	6	0,2
10	1	1	6	0,2
11	1	0	6	0,6
12	1	1	6	0,6
13	2	0	2,6	0,1
14	2	1	2,6	0,1
15	2	0	2,6	0,2
16	2	1	2,6	0,2
17	2	0	2,6	0,3
18	2	1	2,6	0,3
19	2	0	2,6	0,5
20	2	1	2,6	0,5

Figure A.2.1-1 Test Control Table SACIA

A.2.2 Test Control Table PINS2

Test Control Table PINS2 provides the means to study the processing load generated by each individual "event". This was done by placing a single extra "event" in several minor cycles to permit the timing data for that event to be isolated. "Event" in this usage refers to a request for an executive service such as a READ or a WRITE.

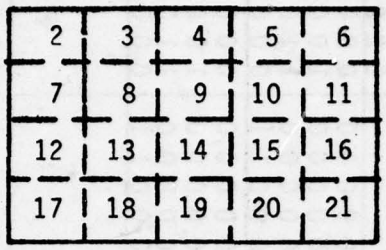
Figure A.2.2-1 shows the background load that is executing during each test phase, the number of extra "events" occurring in each test phase, and the minor cycles (modulo 8) in which these extra "events" are to occur. As always, the first phase is the calibration phase. Phases 2 through 6 examine the effects of adding events in minor cycle 1, while phases 7 through 11 do the same for minor cycle 5 and phases 12 through 16 examine minor cycle 6. Phases 17 through 21 repeat phases 12 through 16 while adding a number of local writes.

Figure A.2.1-2 Processing Load Matrix for Test Control Table SACIA

PARAMETER	MINOR CYCLES 0(a)	1	2	3	4	5(b)	6	7
READS	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 3 4 3 4 3 4 3 4 3 4 3 4 3 4	2 2 2 2 2 2 2 2 2 2 2 2 4 5 4 5 4 5 4 5	1 1 1 1 1 1 1 1 1 1 1 1 2 3 2 3 2 3 2 3	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2 2 2 3 2 3 2 3 2 3 2 3 3 4 3 4 3 4 3 4	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
WRITES (Local)		0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1			0 0 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
WRITES (Global)		0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1				1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2	
SCHEDULE (In other Processor)	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0	0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
CANCEL	0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0	0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0		0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SIGNAL (In other Processor)								
TRIGGER			0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1				0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
SCHEDULE (In Local Processor)								
SIGNAL (In Local Processor)								

(a) BCIU Transfers 2 Compools
(b) BCIU Transfers 4 Compools

Figure A.2.2-2 provides an explicit listing of the number of executive functions being performed in each minor cycle of each test phase of TCT PINS2. In each block, the test phases are arranged in the following pattern:



The calibration phase (phase 1) is not shown in figure A.2.2-2.

			MINOR CYCLES				
TEST PHASE	BACKGROUND LOAD	LOCAL WRITE	TRIGGER	SCHEDULE AND CANCEL	READ AND WRITE	SIGNAL	EXTRA TASK
1	0	0					
2	0	0	1				
3	0	0		1			
4	0	0			1		
5	0	0				1	
6	0	0					1
7	0	0	5				
8	0	0		5			
9	0	0			5		
10	0	0				5	
11	0	0					5
12	0	0	6				
13	0	0		6			
14	0	0			6		
15	0	0				6	
16	0	0					6
17	0	3	6				
18	0	3		6			
19	0	3			6		
20	0	3				6	
21	0	3					6

Figure A.2.2-1 Test Control Table PINS2

Figure A.2.2-2 Processing Load Matrix for Test Control Table PINS2

PARAMETER	MINOR CYCLES							
	0(a)	1	2	3	4	5(b)	6	7
READS	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1	2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 4 4 5 4 4	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
WRITES (Local)			0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1				0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2	
WRITES (Global)		0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1			0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 2 1 1	
SCHEDULE (In other Processor)		0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0				0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0	
CANCEL		0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0				0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0	
SIGNAL (In other Processor)		0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0				0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0	
TRIGGER		1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0				0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0	
SCHEDULE (In Local Processor)		0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0				0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	
SIGNAL (In Local Processor)		0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0				0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	

(a) BCIU Transfers 2 Compools
(b) BCIU Transfers 4 Compools

A.2.3 Test Control Table PINS3

In TCT PINS3, each of the user controlled PINS events (TRIGGER, SCHEDULE and CANCEL, READ and WRITE, SIGNAL, and Extra Task) is studied as a function of the PINS base load. A single occurrence of each event is added to each minor cycle in turn. This permits the study of the executive performance when processing the event in a variety of operating environments. The executive performance is not investigated in minor cycle 7 since the base load in that minor cycle is identical to the base load in minor cycle 3 (see figure 3.1-5). Instead, the eighth test phase for each event calls for the processing of that event in every minor cycle.

Figure A.2.3-1 shows the background load that is executing during each test phase, the number of extra "events" occurring in each test phase, and the minor cycles (modulo 8) in which these extra events are to occur. The first phase is the calibration phase (the base load shown in figure 3.1-5). Phases 2 through 8 examine the TRIGGER statement, phases 9 through 16 study SCHEDULEs and CANCELs, phases 17 through 24 study READs and WRITEs, phases 25 through 32 study SIGNALs, and phases 33 through 40 examine Extra Tasks.

Figure A.2.3-2 provides an explicit listing of the number of executive functions being performed in each minor cycle of each test phase of TCT PINS3. Since there are too many phases to include on a single chart, figure A.2.3-2 has been split into two parts. Figure A.2.3-2a shows the tasks performed in the first 20 phases of TCT PINS3 while figure A.2.3-2b presents the tasks performed in the last 20 phases of TCT PINS3. In the data blocks given in the figures, the test phases are arranged as follows:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

Test Phase Pattern in Figure A.2.3-2a

21	22	23	24	25
26	27	28	29	30
31	32	33	34	35
36	37	38	39	40

Test Phase Pattern in Figure A.2.3-2b

			MINOR CYCLES				
TEST PHASE	LOAD	LOCAL WRITE	TRIGGER	SCHEDULE AND CANCEL	READ AND WRITE	SIGNAL	EXTRA TASK
1	0	0					
2	0	0	0				
3	0	0	1				
4	0	0	2				
5	0	0	3				
6	0	0	4				
7	0	0	5				
8	0	0	6				
9	0	0		0			
10	0	0		1			
11	0	0		2			
12	0	0		3			
13	0	0		4			
14	0	0		5			
15	0	0		6			
16	0	0		0-7			
17	0	0			0		
18	0	0			1		
19	0	0			2		
20	0	0			3		
21	0	0			4		
22	0	0			5		
23	0	0			6		
24	0	0			0-7		
25	0	0				0	
26	0	0				1	
27	0	0				2	
28	0	0				3	
29	0	0				4	
30	0	0				5	
31	0	0				6	
32	0	0				0-7	
33	0	0				0	
34	0	0				1	
35	0	0				2	
36	0	0				3	
37	0	0				4	
38	0	0				5	
39	0	0				6	
40	0	0				0-7	

Figure A.2.3-1 Test Control Table PINS 3

Figure A.2.3-2a Processing Load Matrix, TCT PINS3, First 20 Phases

PARAMETER	MINOR CYCLES							
	0(a)	1	2	3	4	5(b)	6	7
READS	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
WRITES (Local)								
WRITES (Global)	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1			1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	
SCHEDULE (In other Processor)	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
CANCEL	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
SIGNAL (In other Processor)								
TRIGGER	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	
SCHEDULE (In Local Processor)								
SIGNAL (In Local Processor)								

(a) BCIU Transfers 2 Compools
(b) BCIU Transfers 4 Compools

Figure A.2.3-2b Processing Load Matrix, TCT PINS3, Last 20 Phases

PARAMETER	MINOR CYCLES							
	0(a)	1	2	3	4	5(b)	6	7
READS	1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
WRITES (Local)								
WRITES (Global)	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
SCHEDULE (In other Processor)								
CANCEL								
SIGNAL (In other Processor)	0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
TRIGGER								
SCHEDULE (In Local Processor)	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
SIGNAL (In Local Processor)	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

(a) BCIU Transfers 2 Compools
(b) BCIU Transfers 4 Compools

A.2.4 Test Control Table PINS4

Test Control Table PINS4 examines the DAIS executive performance when two of the user controlled requests occur in the same minor cycle. Only the more likely combinations or requests were selected for inclusion in TCT PINS4. An examination of typical applications design requirements indicated that READs and WRITEs, and SIGNALs would generally be the most numerous realtime statements executed in a program. TCT PINS4 combined these events with others in the following test phases:

Base Task	Added Task	Test Phases
READ and WRITE	TRIGGER	6-12
READ and WRITE	SCHEDULE and CANCEL	20-26
READ and WRITE	SIGNAL	34-40
SIGNAL	TRIGGER	13-19
SIGNAL	SCHEDULE and CANCEL	27-33

Figure A.2.4-1 shows the background load that is executing during each test phase, the number of extra "events" occurring in each test phase, and the minor cycles (modulo 8) in which these extra events are to occur. The first five phases in TCT PINS4 are calibration phases: The base load is executed in phase one, while the user controlled extra events are examined separately in phases 2 through 5. Figure A.2.4-2 provides the explicit listing of executive functions in a pattern identical to that in figure A.2.3-2 (the test phase pattern is described in paragraph A.2.3).

			MINOR CYCLES			
TEST PHASE	LOAD	LOCAL WRITE	TRIGGER	SCHEDULE AND CANCEL	READ AND WRITE	SIGNAL
1	0	0				
2	0	0	0			
3	0	0		0		
4	0	0			0	
5	0	0				0
6	0	0	0		0	
7	0	0	1		1	
8	0	0	2		2	
9	0	0	3		3	
10	0	0	4		4	
11	0	0	5		5	
12	0	0	6		6	
13	0	0	0			0
14	0	0	1			1
15	0	0	2			2
16	0	0	3			3
17	0	0	4			4
18	0	0	5			5
19	0	0	6			6
20	0	0		0	0	
21	0	0		1	1	
22	0	0		2	2	
23	0	0		3	3	
24	0	0		4	4	
25	0	0		5	5	
26	0	0		6	6	
27	0	0		0		0
28	0	0		1		1
29	0	0		2		2
30	0	0		3		3
31	0	0		4		4
32	0	0		5		5
33	0	0		6		6
34	0	0			0	0
35	0	0			1	1
36	0	0			2	2
37	0	0			3	3
38	0	0			4	4
39	0	0			5	5
40	0	0			6	6

Figure A.2.4-1 Test Control Table, PINS4

Figure A.2.4-2a Processing Load Matrix, TCT PINS4, First 20 Phases

PARAMETER	MINOR CYCLES							
	0(a)	1	2	3	4	5(b)	6	7
READS	1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2	1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1	2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2	1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1	2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
WRITES (Local)								
WRITES (Global)	0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1	
SCHEDULE (In other Processor)	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1							
CANCEL	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1							
SIGNAL (In other Processor)	0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	
TRIGGER	0 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0	0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0	
SCHEDULE (In Local Processor)								
SIGNAL (In Local Processor)								

(a) BCIU Transfers 2 Compools
(b) BCIU Transfers 4 Compools

Figure A.2.4-2b Processing Load Matrix, TCT PINS4, Last 20 Phases

PARAMETER	MINOR CYCLES							
	0(a)	1	2	3	4	5(b)	6	7
READS	1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1	2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1	2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2	1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1	1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1	1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1	2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
WRITES (Local)								
WRITES (Global)	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0	1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2	
SCHEDULE (In other Processor)	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0	
CANCEL	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0	0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0	0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0	0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0	
SIGNAL (In other Processor)	0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1	
TRIGGER								
SCHEDULE (In Local Processor)								
SIGNAL (In Local Processor)								

(a) BCIU Transfers 2 Compools
(b) BCIU Transfers 4 Compools

Appendix B

Laboratory Configuration

The test facility used to evaluate the performance of the DAIS executive consists of the Boeing DARTS facility (paragraph B.1), GFE supplied by AFAL (paragraph B.2), the DEC-10 facility at the University of Washington (paragraph B.3), the software that was supplied by AFAL to be evaluated (paragraph B.4), and the software developed by Boeing to evaluate the DAIS executive (paragraph B.5).

B.1 Boeing DARTS Facility

The Boeing DARTS facility contains many computers, programmable bus control interface units, computer peripheral devices, test equipment, and support software. Only a portion of the equipment in DARTS was used in the executive evaluation program. Those elements of the DARTS facility used are shown in figure B.1-1. A description of the elements is given below.

B.1.1 PDP 11/40

The PDP 11/40 was used to control the AN/AYK-15 processing load during test operation. The PDP 11/40 also provided control of online recording (tape or disk), realtime display (CRT), and offline non-realtime display.

B.1.2 Harris/6 - DARTS

The Harris/6 computer provided simulated realtime sensor inputs, and accepted control and display and sensor correction outputs. The Harris/6 peripheral devices included a CRT for realtime display, a magnetic tape transport for data recording and offline data analysis, a printer and card reader for offline program generation and program control, and a disk for system residency.

B.1.3 DARTS Bus Control Interface Units (BCIU)

These Boeing-built devices provided the MIL-STD-1553 multiplex bus interface within the DARTS facility. Two BCIU's were used, one for the PDP 11/40 interface and one for the Harris/6 interface. The DARTS BCIU's are programmable and can act as bus controllers (master or slave) or a DAIS defined universal remote terminal.

B.1.4 Modest Control and Display Units (MCADU)

The Boeing-built MCADU's were connected to the Super Control and Display Unit (SCADU) port on the AN/AYK-15 processors. They provided a capability for realtime data recording by passing data from the AN/AYK-15 to the PDP 11/40. The MCADU's operate on an address and instruction trigger and when both are satisfied, the MCADU will read data from an internal AN/AYK-15 data bus, buffer this data and signal the PDP 11/40 that data is available. The MCADU is housed in the PDP 11/40 BCIU equipment rack.

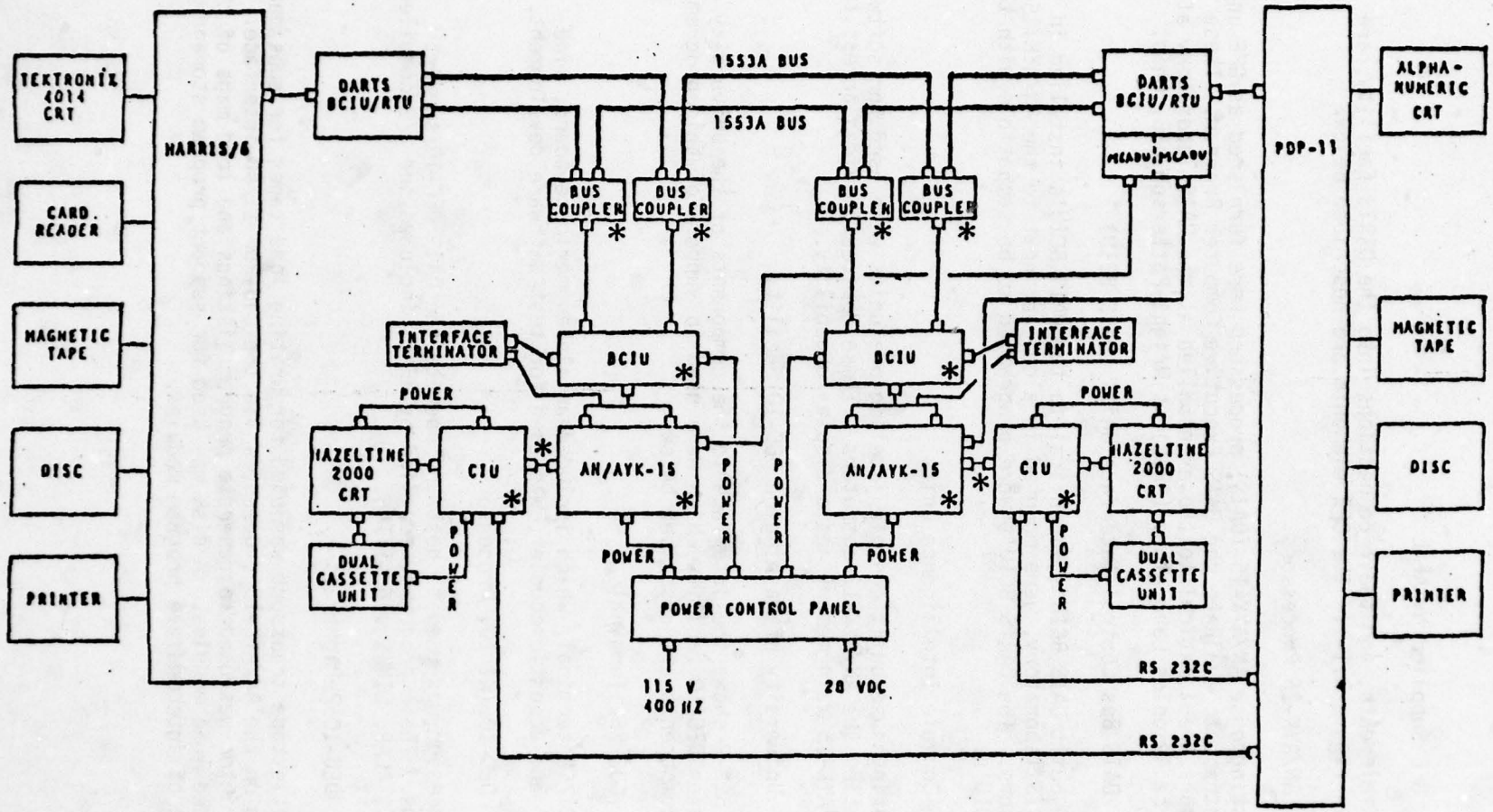


Figure B.1-1 DARTS Facility Hardware

B.2 GFE Supplied by AFAL

The GFE elements, and their connections into the DARTS facility, are shown in figure B.1-1. The GFE elements are described below.

B.2.1 AN/AYK-15 Processors

Two Westinghouse AN/AYK-15 (DAIS) processors were furnished as GFE under the contract to evaluate the DAIS Executive Computer Program. These processors are identical to those installed at the DAIS laboratory at the Air Force Avionics Laboratory (AFAL) at Wright-Patterson AFB, Ohio.

B.2.2 DAIS Bus Control Interface Units (DAIS BCIU)

Two IBM-built DAIS BCIU's, identical to the DAIS BCIU's installed in the AFAL DAIS laboratory, were mounted in a cabinet next to the AN/AYK-15 processors. The DARTS BCIU's were programmed to be compatible with the DAIS BCIU's.

B.2.3 Console Intelligence Units

The Westinghouse-built console intelligence units were used for software checkout on the AN/AYK-15 computers. These devices provided access to the AN/AYK-15 main memory and a display capability.

B.3 University of Washington DEC-10 Facility

Figure B.2-2 shows the interface to the components of the University of Washington DEC-10 facility which were used in support of this program. These components are described below.

B.3.1 CDC 752 Terminal

The CDC 752 terminal, which included an alphanumeric keyboard, line printer and acoustic coupler, was used for test software development.

B.3.2 DEC-10 (KI 10) Processor

The processor was used to host and operate the DAIS DEC-10 support programs. These programs, supplied by AFAL, included the J73 compiler, PALEFAC, ALAP, LINKS and ASYTRN.

B.3.3 DEC-10 Peripherals

A magnetic tape transport was used for writing load tapes for subsequent loading on the AN/AYK-15 processors via the CIU-PDP 11/40 interface. A line printer was used to generate program listings and load maps of the generated load modules. A disk was used for support program storage and storage of intermediate program modules.

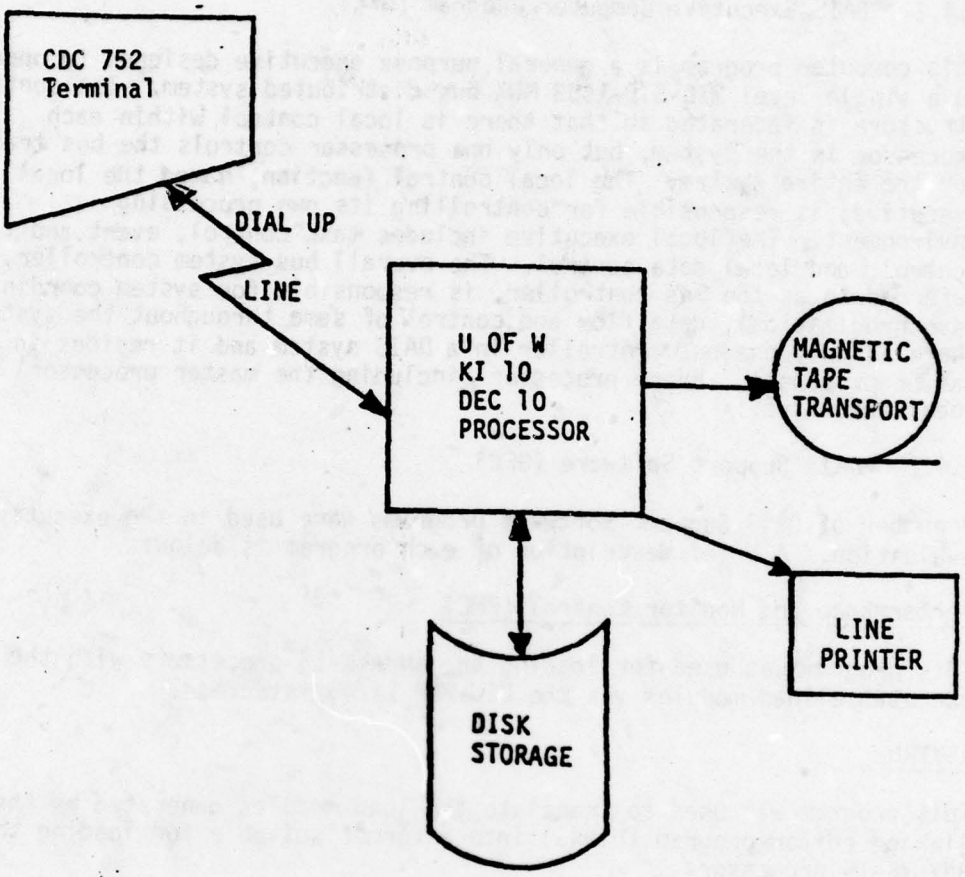


Figure B.2-1 University of Washington DEC-10 Facility

B.4 GFE Software

The GFE software programs to be evaluated were the DAIS executive and the DAIS Support Software used in the development of standard application software. These programs are briefly described below.

B.4.1 DAIS Executive Computer Program (GFE)

This computer program is a general purpose executive designed to operate on a single level MIL-STD-1553 MUX bus distributed system. The control structure is federated so that there is local control within each processor in the system, but only one processor controls the bus traffic for the entire system. The local control function, named the local executive, is responsible for controlling its own processing environment. The local executive includes task control, event and timing control, and local data control. The overall bus system controller, referred to as the bus controller, is responsible for system coordination (synchronization), data flow and control of same throughout the system. There is only one bus controller in a DAIS system and it resides in the master processor. Every processor (including the master processor) has a local executive.

B.4.2 DAIS Support Software (GFE)

A number of DAIS Support Software programs were used in the executive evaluation. A brief description of each program is below:

Performance and Monitor Control (PMC)

This program was used for loading the AN/AYK-15 processors with the executable load modules via the CIU-PDP 11/40 interface.

ASYTRN

This program was used to translate the load modules generated by the linkage editor program (LINKS) into a format suitable for loading the AN/AYK-15 processors.

J73/I Compiler

This program was used to read and interpret source programs written in J73/I into object code (machine language).

PALEFAC

This DAIS support program was used to generate the tables which provide the interface between the DAIS Executive Computer Program and the operational applications software.

PALEFAC Preprocessor

This program was used to read programs written in J73/I source language and extract data to be used by the PALEFAC program to generate the necessary interface tables.

LINKS

This program was used as the linkage editor to process the output of the J73/I compiler and ALAP into executable load modules. Processing primarily consisted of resolving external references in other compiled units within the overall load modules and assigning absolute main memory addresses.

ALAP

This program was used to process source programs written in assembler language for the Westinghouse AN/AYK-15 processors. The output of this program was suitable for processing by the LINKS program.

B.5 Boeing Developed Test Software

Six computer programs were developed by Boeing to assist in the evaluation of the DAIS executive. For the Phase I parametric evaluation, an "executive test program" (paragraph B.5.1) was developed for counting the number of executive tasks executed and measuring the time required for each. A programmable applications program called the Pseudo-Integrated Navigation System (paragraph B.5.2) was also developed for creating predictable tasks for the executive. Test control software (paragraph B.5.3) for the DARTS facility test control and data analysis software (paragraph B.5.4) were also developed and were used in both the Phase I parametric test and the Phase II validation test. Integrated navigation system software (paragraph B.5.5) developed by Boeing for a medium STOL transport aircraft was used to verify the test results from Phase I. Sensor simulation programs resident in the Harris/6 "DARTS Environmental Control System Simulation" (paragraph B.5.6) were used to simulate aircraft and sensor inputs to the integrated navigation software. A description of the Boeing developed software is below.

B.5.1 Executive Test Program

This program provides hooks for instrumenting the DAIS executive to branch to the performance data gathering programs. The performance data gathering programs selectively gather performance data on the DAIS executive and make this data available to the MCADU to ensure that the performance data is recorded.

B.5.2 Pseudo-Integrated Navigation System (PINS)

This program can simulate the processing load of a real avionics program. Through PINS' various levels of message transmission on the MIL-STD-1553 MUX bus and executive service requests can be programmed to simulate virtually any avionic software function. The program was designed so that message levels, requests for executive services and processing resource demands could be controlled in realtime. This program is described in detail in "DAIS Executive Evaluation Software Design," Attachment #2, CDRL #1.

The primary use for this program was during the Phase I parametric study because different levels of activity could be controlled and performance of the DAIS Executive Computer Programs could be examined at these controlled levels.

B.5.3 DARTS Facility Support Software

This software provides a number of functions associated with testing and control. The test control function provides the capability for control messages to be transmitted from the PDP 11/40 to an AN/AYK-15 via the MUX bus in realtime. The bus monitor function allows selective bus traffic monitoring by terminal address, subaddress and type (such as send or receive mode code). This function also provides for realtime display via a DARTS CRT. The data recording function reads performance data from the MCADU's and sends it to the magnetic tape transport for recording.

B.5.4 Data Analysis Program

This program reads the performance data recorded by the executive evaluation software via the MCADU-PDP 11/40 interface and produces statistical data and reports. The program contains the capability to select the data to be operated on, based upon input parameters. One of the input parameters is the "DAIS Executive Computer Program" time (i.e., minor cycle).

B.5.5 DARTS Integrated Navigation System (DINS)

This is an avionics program designed from the requirements of a tactical transport aircraft. The prime function of the program is integrated navigation using a strapdown IMU, GPS receiver models, and a Kalman filter smoothing and prediction algorithm. In addition to the integrated navigation function of the program, limited controls and display, and simulated computed air release point algorithms were mechanized. DINS was designed to operate under control of the DAIS executive in a single level distributed MUX data bus system just as the PINS program was.

B.5.6 DARTS Environmental Control System Simulation

This program contains models for GPS receiver, air data computer, strapdown IMU and associated controls and displays. These models were used for environmental simulation and in the laboratory provide simulated sensor inputs to the navigation program. DECSS accepts sensor corrections supplied by the navigation program and applies these corrections in the models as if they were being applied to live sensors. The DECSS program responds to up to 31 terminal addresses for simulating multiple sensors and/or remote terminals.

Appendix C

Phase I Data Samples

This appendix contains samples of the data collected during Phase I of the DAIS executive study. The samples consist of statistical summaries produced by the offline-analysis program from magnetic tapes recorded during Phase I tests. A description of the method used to collect the data is provided in paragraph 3.1.3. The format of the data contained in the statistical summaries is described in paragraph 3.1.5.

Three sets of sample data are provided from the Phase I tests. Each set consists of statistical output from nine passes through the magnetic tape recorded during the test. The first pass combines all data collected during each minor cycle. The succeeding eight passes each examine one minor cycle (modulo 8) in turn by extracting and compiling statistics on just the data collected in that minor cycle.

The first set of data is from an Interrupt Service Overhead (ISO) test performed on the master processor. This particular test was designed to concentrate on requests received from the remote processor; the master configurator was disabled to prevent any master processor response to Test Control Table (TCT) inputs. TCT PINS3 (described in Appendix A) was used to control the asynchronous executive requests serviced in the remote processor. The interrupts for which data were collected are identified by the last number provided in the event type heading on the listings.

The second set of data is from an ISO test performed on the remote processor. This test was designed to concentrate on requests originating in the remote processor; the master configurator was disabled to prevent any master processor response to the TCT inputs. TCT PINS4 (described in Appendix A) was used to control the asynchronous executive requests serviced in the remote processor.

The final set of data is from a Transmission Delay Time Test (TDT). Since the TDT test only monitors executive service requests originating in the remote processor, the master configurator was disabled during this test. TCT PINS3 (described in Appendix A) was used to control the asynchronous inputs for this test. This TCT generates SCHEDULE requests only in test phases 9 through 16; therefore, TDT data was collected only during these test phases.

Appendix D

Phase II Data Samples

This appendix contains samples of the data collected during Phase II of the DAIS executive study. The samples consist of statistical summaries produced by the offline analysis program from magnetic tapes recorded during Phase II tests. A description of the method used to collect the data is provided in paragraph 3.2.3. The format of the data contained in the statistical summaries is described in paragraph 3.1.5.

Two sets of sample data are provided from the Phase II tests. Each set consists of statistical output from nine passes through the magnetic tape recorded during the test. The first pass combines all data collected during each minor cycle. The succeeding eight passes each examine one minor cycle (modulo 8) in turn by extracting and compiling statistics on just the data collected in that minor cycle.

The first set of data is from an ISO test performed on the master processor. The data cover four complete Kalman filter cycles, each of which lasts six seconds. The high processing time observed in the remote processor reflects the large number of matrix manipulations occurring in the navigation system. The load is highest in the first three seconds of each six second sequence since the Kalman filter is processed then. The interrupts for which data are collected are identified by the last number provided in the event type heading on the listings.

The second set of data is from an ISO test performed on the remote processor. The master processing times shown in the listing are meaningless because the software clock that measured processing time was not instrumented in this test.