AFATL-TR-88-62, VOL III

Common Ada Missile Packages-Phase 2 (CAMP-2)

Volume III. CAMP Armonics Benchmarks

S Cohen
T Taylor

McDONNELL DOUGLAS ASTRONAUTICS COMPANY
P O BOX 516
ST LOUIS, MO 63166

NOVEMBER 1988

FINAL REPORT FOR PERIOD SEPTEMBER 1985-MARCH 1988

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▪ United States Air Force ▪ Eglin Air Force Base, Florida

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS CRITICAL TECHNOLOGY
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Distribution authorized to U.S. Government Agencies and their contractors; ~~this report contains test and evaluation.~~ (OVER)
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFATL-TR-88-62, Volume III

6a. NAME OF PERFORMING ORGANIZATION McDonnell Douglas Astronautics Company	6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Aeromechanics Division Guidance and Control Branch
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 516 St Louis MO 63166		7b. ADDRESS (City, State, and ZIP Code) Air Force Armament Laboratory Eglin Air Force Base, Florida 32542-5434
8a. NAME OF FUNDING/SPONSORING ORGANIZATION STARS Joint Program Office	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F08635-86-C-0025

8c. ADDRESS (City, State, and ZIP Code)
Room 3D139 (1211 Fern St)
The Pentagon
Washington DC 20301-3081

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO	WORK UNIT ACCESSION NO.
63756D	921D	GT	02

11. TITLE (Include Security Classification) Common Ada Missile Packages-Phase 2 (CAMP-2), Volume III: CAMP Armonics Benchmarks

12. PERSONAL AUTHOR(S)
S. Cohen and T. Taylor

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM Sep 85 TO Mar 88	November 1988	70

16. SUPPLEMENTARY NOTATION

Availability of this report is specified on verso of front cover. (OVER)

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Reusable Software, Missile Software, Software Generators, Ada parts, Composition, Systems, Software Parts

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

The CAMP project, primarily funded ty the STARS Joint Program Office, sponsored by the Air Force Armament Laboratory, and performed by McDonnell Douglas, has taken a pragmatic approach to demonstrating the feasibility and utility of the concept of software reuse for real-time embedded missile systems. CAMP products include: 452 operational flight software parts in Ada for tactical missiles, and a prototype parts engineering system to support parts identification, cataloging and construction. In order to demonstrate the value of the reuse concept, a missile subsystem was built using the CAMP parts. Results indicate a significant increase in software productivity when developing systems using parts, Ada, modern software engineering practice, robust software tools, and knowledgeable software engineers.

This report is documented in three volumes: Volume I - CAMP Parts and Parts Composition System, Volume II - 11th Missile Demonstration, and Volume III - CAMP Armonics Benchmarks.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Christine M. Anderson	22b. TELEPHONE (Include Area Code) (904) 882-2961	22c. OFFICE SYMBOL AFATL/FXG

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

distribution limitation applied March 1988. Other requests for this document must be referred to the Air Force Armament Laboratory (FXG), Eglin Air Force Base, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

TRADEMARKS

The following table lists the trademarks used throughout this document:

TRADEMARK	TRADEMARK OF
ACT	Advanced Computer Techniques
ART	Inference Corporation
ART Studio	Inference Corporation
CMS	Digital Equipment Corporation
DEC	Digital Equipment Corporation
Mikros	Mikros, Inc.
Oracle	Oracle Corporation
Scribe	Scribe Systems
Symbolics	Symbolics, Inc.
Symbolics 3620	Symbolics, Inc.
TLD	TLD Systems Ltd
VAX	Digital Equipment Corporation
VMS	Digital Equipment Corporation

PREFACE

This report describes the work performed, the results obtained, and the conclusions reached during the Common Ada Missile Packages Phase-2 (CAMP-2) contract (F08635-86-C-0025). This work was performed by the Software and Information Systems Department of the McDonnell Douglas Astronautics Company, St. Louis, Missouri (MDAC-STL), and was sponsored by the United States Air Force Armament Laboratory (FXG) at Eglin Air Force Base, Florida. This contract was performed between September 1985, and March 1988.

The MDAC-STL CAMP program manager was:

Dr. Daniel G. McNicholl
Technology Branch
Software and Information Systems Department
McDonnell Douglas Astronautics Company
P.O. Box 516
St. Louis, Missouri 63166

The AFATL CAMP program manager was:

Christine M. Anderson
Guidance and Control Branch
Aeromechanics Division
Air Force Armament Laboratory
Eglin Air Force Base, Florida 32542-5434

This report consists of three volumes. Volume I contains information on the development of the CAMP parts and the Parts Composition System. Volume II contains the results of the 11th Missile Application development. Volume III contains the results of the CAMP Armonics Benchmarks Suite development.

Commercial hardware and software products mentioned in this report are sometimes identified by manufacturer or brand name. Such mention is necessary for an understanding of the R & D effort, but does not constitute endorsement of these items by the U.S. Government.

ACKNOWLEDGEMENT

Special thanks to the Armament Division Deputy for Armament Control Office; to the Software Technology for Adaptable, Reliable Systems (STARS) Joint Program Office; to the Ada Joint Program Office (AJPO); and to the Air Force Electronic Systems Division, Computer Resource Management Technology Program Office for their support of this project.

TRADEMARKS

The following table lists the trademarks used throughout this document:

TRADEMARK	TRADEMARK OF
ACT	Advanced Computer Techniques
ART	Inference Corporation
ART Studio	Inference Corporation
CMS	Digital Equipment Corporation
DEC	Digital Equipment Corporation
Mikros	Mikros, Inc.
Oracle	Oracle Corporation
Scribe	Scribe Systems
Symbolics	Symbolics, Inc.
Symbolics 3620	Symbolics, Inc.
TLD	TLD Systems Ltd
VAX	Digital Equipment Corporation
VMS	Digital Equipment Corporation

Table of Contents

Table of Contents (CONCLUDED)

List of Figures

List of Tables

List of Acronyms

ACS	Ada Compilation System
ACVC	Ada Compiler Validation Capability
AdaJUG	Ada/Jovial Users Group
ADL	Ada Design Language
AFATL	Air Force Armament Laboratory
AFB	Air Force Base
AI	Artificial Intelligence
AJPO	Ada Joint Program Office
AMPEE	Ada Missile Parts Engineering Expert (System)
AMRAAM	Advanced Medium Range Air-to-Air Missile
ANSI	American National Standards Institude
APSE	Ada Programming Support Environment
Armonics	Armament Electronics
ART	Automated Reasoning Tool
ASCII	American Standard Code for Information Interchange
BC	Bus Controller
BDT	Basic Data Types
BIM	Bus Interface Module
CAD/CAM	Computer-Aid Design/Computer-Aided Manufacturing
CAMP	Common Ada Missile Packages
CCCB	Configuration Change Control Board
CDRL	Contractual Data Requirements List
CMS	Code Management System
ConvFactors	Conversion_Factors (TLCSC)
CPDS	Computer Program Development Specification
CPPS	Computer Program Product Specification
CSC	Computer Software Component
CSCI	Computer Software Configuration Item
CVMA	Coordinate_Vector_Matrix_Algebra (TLCSC)
DACS	Defense Analysis Center for Software
DBMS	Data Base Management System
DCL	DIGITAL Command Language
DDD	Detailed Design Document
DEC	Digital Equipment Corporation
DMA	Direct Memory Access
DoD	Department of Defense

DoD-STD	Department of Defense Standard
DPSS	Digital Processing Subsystem
DSR	Digital Standard Runoff
DTM	DEC /Test Manager
FMS	Forms Management System
FORTRAN	FORmula TRANslation
GPMath	General_Purpose_Math (TLCSC)
HOL	Higher-Order Language
Hr	Hour
I/O	Input/Output
ISA	Inertial Sensor Assembly
JOVIAL	Jules Own Version of International Algebraic Language
LISP	List Processing (language)
LLCSC	Lower Level Computer Software Component
LOC	Lines of Code
MDAC	McDonnell Douglas Astronautics Company
MDAC-HB	McDonnell Douglas Astronautics Company - Huntington Beach
MDAC-STL	McDonnell Douglas Astronautics Company - St. Louis
MDC	McDonnell Douglas Corporation
MIL-STD	Military Standard
MRASM	Medium Range Air-to-Surface Missile
NM	Nautical Miles
NPNav	North_Pointing_Navigation_Parts (TLCSC)
OCU	Operator Control Unit
Opns	Operations
PC	Personal Computer
PCS	Parts Composition System
PDL	Program Design Language
R&D	Research and Development
RT	Remote Terminal
RTE	Real-Time Embedded
SDF	Software Development File
SDI	Strategic Defense Initiative
SDN	Software Development Notebook
SDR	Software Discrepancy Report
SEAFAC	System Engineering Avionics Facility
SEI	Software Engineering Institute

SEP/SCP	Software Enhancement Proposal/Software Change Proposal
SIGAda	Special Interest Group on Ada
SRS	Software Requirements Specification
STARS	Software Technology for Adaptable, Reliable Systems
stmt	statement
SURMOS	Start-Up Real-time Multi-tasking Operating System
TLCSC	Top-Level Computer Software Component
TLDD	Top-Level Design Document
UnivConst	Universal_Constants (TLCSC)
VAX	Virtual Address Extension
VMS	Virtual Memory System
WGS72	World Geodetic System, 1972

SECTION I

INTRODUCTION

1. IDENTIFICATION

The CAMP Armonics Benchmark Suite facilitates the evaluation of Ada software engineering environments and microprocessors intended for use in armonics[1] applications. The suite features both compilation and execution benchmarks to measure the capabilities of compiler/run-time systems. All benchmarks in this suite are portable and will permit comparisons to be made between widely different Ada systems.

This volume identifies the benchmarks and benchmark drivers, and suggests techniques for applying the Benchmark Suite. In addition, the structure, purpose, and methodology of the suite are explained to familiarize readers with the suite and to facilitate the evaluation of the suite by engineers. For those interested in using the benchmarks, a guide is provided in Section V. Appendix A contains data collected in the process of running the Benchmark Suite.

2. SYSTEM OVERVIEW

This Armonics Benchmark Suite serves a dual purpose: it offers a means for assessing the performance of CAMP parts and, at the same time, provides support for evaluating the suitability of compiler systems and their target machines to armonics applications.

Ada compiler performance is tested by a series of compilations, based on CAMP packages, which require a compiler to process complex uses of Ada generic units. These advanced (but standard) Ada features are used heavily in the CAMP parts and are central to the development and use of reusable software.

Other benchmarks of the suite are targeted primarily at run-time performance issues such as storage requirements, execution time, and computational accuracy. These benchmarks consist of a selection of CAMP parts which have been chosen as representative of the needs of armonics applications. Testing, using these benchmarks, is facilitated by embedding the benchmarks within portable drivers, written in Ada. Effectively, this allows the benchmarks to run themselves.

The Benchmark Suite can support a number of benchmarking scenarios:

- A project wishes to evaluate compilers for use in the development of a reusable parts library. The suite provides test code for evaluating compiler/linker systems.
- A compiler developer wants to measure the performance of his compiler/run-time system against an established standard. A group of benchmarks documented in this volume provides a standard for comparison between different systems.

[1]armament electronics

- An armonics application needs data on the memory utilization and timing efficiency of several compilers in order to select an appropriate compiler for a new project. The benchmarks provide opportunities for measuring these features of a given compiler.
- A potential user of CAMP parts wants specific perfonnance data on the parts. The Benchmark Suite gives a user the ability to measure performance for a selected group of parts on varying architectures.
- A scientific application requires transcendental functions of known accuracy on a specific system and is considering the CAMP polynomial parts. The benchmarks supply data on the scientific functions of the CAMP Polynomials package.

The Benchmark Suite is supplemented by a set of procedures, encoded in DEC VAX Digital Command Language (DCL). By performing these procedures (or their equivalents on other operating systems), an engineer may install, compile, and run the various benchmarks efficiently. Details on the use of the Benchmark Suite and its command procedure environment are discussed in Section V.

3. VOLUME OVERVIEW

This report contains five main sections:

1. Introduction: Introduces the CAMP Benchmark Suite and this volume.
2. Benchmark Definitions: Explains the system of classes and levels by which the benchmarks are characterized. This section also introduces key terms and gives a tabular summary of benchmarks contained in the suite.
3. Purpose and Design: Discusses the procedure used to run each of the benchmarks and gives information about their structure and scope. For each benchmark, this section provides the following information where applicable:
   - Benchmark name
   - Compilation structure
   - Benchmark driver design
   - Benchmark inputs
   - Benchmark correct outputs
   - Data to be recorded
   - Methods for recording data
4. Methodology: Gives the overall methodology used to construct the Benchmark Suite in terms of portability, validity, and usability.
5. Using the Benchmarks: Explains how to use the benchmarks on a project. Emphasis is placed on making use of the suite command procedure environment to facilitate benchmarking.

SECTION II

BENCHMARK DEFINITIONS

1. BENCHMARK LEVELS

The CAMP Armonics Benchmark Suite supports benchmarking at three hierarchical levels:

- TLCSC benchmarks: complete operational subsystems;
- LLCSC benchmarks: sequentially driven calls to integrated CAMP parts;
- Unit benchmarks: benchmarks of individual parts. (This level is generally reserved for the benchmarks derived from CAMP polynomial parts .)

2. BENCHMARK CLASSES

The benchmark suite is functionally partitioned into three classes. The *compilation benchmarks* test the ability of an Ada compiler to process source code typical of armonics applications and reusable software. Benchmarks based on the CAMP Polynomials scientific function package are called the *polynomial benchmarks*. Finally, the benchmarks developed from CAMP higher-level armonics parts are referred to as *integrated execution benchmarks*. The following subsections define the three classes of benchmarks in greater detail.

a. Compilation Benchmark Class

The compilation benchmarks test an Ada compiler's ability to process reusable software. The benchmarks concentrate on the complex syntax and semantics of several Ada armonics-oriented implementations using CAMP parts. These implementations are skeletal in that they do not actually implement an armonics subsystem but merely collect the necessary CAMP parts via generic instantiation. The instantiated parts *are* invoked in the benchmark code although the run-time effects of the invocations are not within the designed scope of testing. The compilation benchmarks are valid tests of a compiler only up to (and including) the linking phase.

b. Polynomial Benchmark Class

The Benchmark Suite includes benchmarks based on the CAMP Polynomial parts (part number P688). These parts cover a range of basic mathematical functions, and provide a variety of techniques for obtaining results. For each benchmark, the benchmark drivers obtain both execution time data and function argument-result pairs. In addition, compilation and linkage editing of the polynomial benchmarks afford an opportunity to collect object code size data on all of the functions of the Polynomials package.

A software tool provided with the Benchmark Suite performs accuracy analysis and generates reports for the polynomial benchmarks. This tool takes the output produced by the benchmarks and generates a document incorporating time-consumption data and function-result accuracy measurement. The following information is provided by the tool:

- "Truth values" for each function over that function's benchmarked domain;
- Absolute error in the result of each argument-result pair
- Relative error in the result of each pair
- Maximum relative error tracking over the argument domain
- Maximum absolute and relative error over the argument domain
- Root-mean-square relative error over the argument domain

c. Integrated Execution Benchmark Class

The integrated execution benchmarks test aggregations of CAMP armonics parts. These benchmarks concentrate on three of the major operational functions supported by the CAMP parts:

- Waypoint steering
- Navigation
- Kalman filter

In the waypoint steering and navigation cases above, data is gathered on CAMP parts in the context of an armonics application. This method has the virtue of testing the parts in the kinds of programs in which they will actually be applied. The benchmark based on the CAMP Kalman filter parts provides data on these parts as they operate in a unit testing environment. This method permits the full inclusion of all subprograms in the CAMP Kalman filter subsystem TLCSCs.

Output data from the integrated execution benchmark drivers consists of timing and result data on the benchmark subprograms. The timing data characterizes the execution time required to make a single call to the benchmark subprogram. The result data from the subprogram may be compared with the standard data supplied by CAMP as part of the Benchmark Suite. This comparison allows the engineer performing the benchmarks to spot errors and inaccuracies in run-time data processing on his system.

d. Summary

Table 1 summarizes the benchmarks in the CAMP Armonics Benchmark Suite. For each benchmark, the table provides the following information:

- Benchmark name
- Benchmark number: a unique number for each set of benchmarks, corresponding to Section III of this document. This number gives the subsection and the paragraph of Section III where the benchmarks are described. In the case of the polynomial benchmarks, only the subsection number is applicable. The paragraph number tabulated for the polynomial benchmarks is only for serialization.
- Level: TLCSC (T), LLCSC (L), or Unit (U) as defined above
- Class: Compilation (C), Polynomial (P), or Integrated Execution (I) as defined above
- Objective: the objective of the benchmark
- Description: a description of the TLCSCs used in the benchmark (for the polynomial benchmarks, the description lists the polynomial expansion algorithm tested)
- Data to be recorded: summary of data values generated by running the benchmark and recorded in Appendix A of this report

TABLE 1. CAMP ARMONICS BENCHMARK SUMMARY

(1 OF 2)

Benchmark Name	No.	Lev.	Cls.	Objective	Description	Data to Record
Compilation 1	2.1	L	C	Test compilability of parts needed in North Pointing Navigation.	Packages compiled: N_P_Nav_Parts, Polynomial_Parts, General_Purpose_Math, Coord_Vector_Matrix_Alg, Standard_Trig, Basic_Data_Types, Conversion_Factors, WGS72 (Metric), WGS72 (Unitless), Universal_Constants	Object code size. Successful compile. Compilation time.
Compilation 2	2.2	L	C	Test compilability of parts needed in Waypoint Steering.	Packages compiled: Waypoint_Steering, Geometric_Operations, Coord_Vector_Matrix_Alg, Polynomial_Parts, General_Purpose_Math, Standard_Trig, Basic_Data_Types, Conversion_Factors, WGS72 (Metric), Universal_Constants	Object code size. Successful compile. Compilation time.
Compilation 3	2.3	L	C	Test compilability of parts needed in Kalman Filter.	Packages compiled: Kalm_Filter_Compt_H_Parts, Kalm_Filter_Common_Parts, Polynomial_Parts, General_Purpose_Math, Kalman_Data_Types, General_Vector_Matrix_Alg	Object code size. Successful compile. Compilation time.
Integrated Execution 1	3.1	T	I	Test execution efficiency of a guidance computation implementation.	Packages tested: Waypoint_Steering, Signal_Processing	Execution time. Code size. Result data.
Integrated Execution 2	3.2	T	I	Test execution efficiency of a navigation operations implementation.	Packages tested: Comm_Navigation_Parts, Direction_Cosine_Matrix General_Purpose_Math, General_Vector_Matrix_Alg, Wander_Az_Nav_Parts	Execution time. Code size. Result data.
Integrated Execution 3	3.3	T	I	Test execution efficiency of a Kalman Filter implementation.	Packages tested: Abstract_Data_Structs, Kalm_Filter_Common_Parts, Kalm_Filter_Compt_H_Parts, Kalm_Filter_Compx_H_Parts,	Execution time. Code size. Result data.
Sine Execution	4.1	U	P	Test execution efficiency and result precision of sine function.	Methods tested are: Taylor Series, Modified Taylor Series, Hastings Algorithm, Chebyshev Polynomial, System Functions	Execution time. Code size. Result Data
Cosine Execution	4.2	U	P	Test execution efficiency and result precision of cosine function.	Methods tested are: Taylor Series, Modified Taylor Series, Hastings Algorithm, Hart Algorithm, System Functions	Execution time. Code size. Result Data

TABLE 1. CAMP ARMONICS BENCHMARK SUMMARY (CONCLUDED)

Benchmark Name	No.	Lev.	Cls.	Objective	Description	Data to Record
Tangent Execution	4.3	U	P	Test execution efficiency and result precision of tangent function.	Methods tested are: Taylor Series, Modified Taylor Series, Hastings Algorithm, System Functions	Execution time. Code size. Result Data
Arcsine Execution	4.4	U	P	Test execution efficiency and result precision of arcsine function.	Methods tested are: Taylor Series, Fike Semicircle, System Functions	Execution time. Code size. Result Data
Arccosine Execution	4.5	U	P	Test execution efficiency and result precision of arccosine function.	Methods tested are: Taylor Series, Fike Semicircle, System Functions	Execution time. Code size. Result Data
Arctangent Execution	4.6	U	P	Test execution efficiency and result precision of arctangent function.	Methods tested are: Taylor Series, Continued Fraction, Hastings Algorithm, System Functions	Execution time. Code size. Result Data
Square Root Execution	4.7	U	P	Test execution efficiency and result precision of square root function.	Methods tested are: Newton-Raphson Modified Newton-Raphson	Execution time. Code size. Result Data
Log 10 Execution	4.8	U	P	Test execution efficiency and result precision of log 10 function.	Methods tested are: Taylor Series, Cody-Waite, System Functions	Execution time. Code size. Result Data
Log N Execution	4.9	U	P	Test execution efficiency and result precision of log n function.	Methods tested are: Taylor Series, Cody-Waite, System Functions	Execution time. Code size. Result Data
Natural Log Execution	4.10	U	P	Test execution efficiency and result precision of natural log function.	Methods tested are: Taylor Series, Cody-Waite	Execution time. Code size. Result Data

SECTION III

PURPOSE AND DESIGN

1. GENERAL REQUIREMENTS

The CAMP Armonics Benchmark Suite meets the following general requirements:

- Utilizes CAMP parts in structures which simulate their actual use in typical user applications
- Utilizes test data modeled on typical user application data
- Helps assess Ada compilation capabilities, object code size, execution time, and output results
- Permits comparison between a variety of host/target combinations using different Ada compiler/run-time systems
- Allows modification to meet specific needs of future users
- Exhibits high portability
- Is highly automated

a. Identifying Ada Compiler Inadequacies

One problem faced during the development of the CAMP parts was the inability of some Ada compilers to process complex generic units. This is important because Ada generic units play a pivotal role not only in the future development of reusable software, but also in the application of that software. In order to identify Ada compiler inadequacies in the area of reusable software the CAMP benchmarks provide Ada source code benchmarks which heavily utilize Ada generic units.

The compilation benchmarks of the Armonics Benchmark suite go beyond the limited scope of testing in the official Ada Compiler Validation Capability (ACVC) tests. While the ACVC tests demonstrate conformance to the Ada language specification, the effect of *combining* language features in complex ways is not sufficiently addressed. The CAMP compilation benchmarks attempt to bridge the gap between the objectives of the ACVC tests and the necessities of complex software applications. It is believed at this point that very few ACVC-validated Ada compilers will, in fact, correctly handle the CAMP compilation benchmarks.

b. Testing Calculation Accuracies

The CAMP parts, including those selected as benchmarks, consist of portable Ada source code. However, certain aspects of the run-time performance of the parts may still vary from system to system. The accuracies of numeric computations, for instance, are guaranteed by the Ada language definition to meet the minimum requirements specified in the software, but, this does not mean that different compiler implementations of Ada will handle numeric computations in the same way. A compiler is free both to provide *more* accuracy than is requested by application software, and to support *less* accuracy based on the limitations of the target machine. For this reason, the results of calculations performed by portable software may not themselves be portable. Differences in numeric accuracies and range limits in Ada systems introduce the possibility of unanticipated error in extensive calculations. This factor must be considered by potential users of the CAMP parts as it would have to be by users of any software (or hardware) product.

The two classes of execution benchmarks (polynomial and integrated execution) in the Armonics Benchmark Suite address the issue of varying computational accuracies in different Ada systems. They provide a standard means of generating data from the kinds of complex calculations involved in armonics applications.

c. Testing Time and Space Performance

An important performance factor in real-time embedded (RTE) environments is space and time efficiency: Software must be kept small because hardware must be kept small in RTE systems; software must also operate efficiently because of the throughput requirements of real-time processing. The execution benchmarks of the Benchmark Suite support execution-time testing of CAMP parts as they operate on various Ada compiler/target machine systems. Selected CAMP parts make up the benchmarks which cover operations common to many armonics applications.

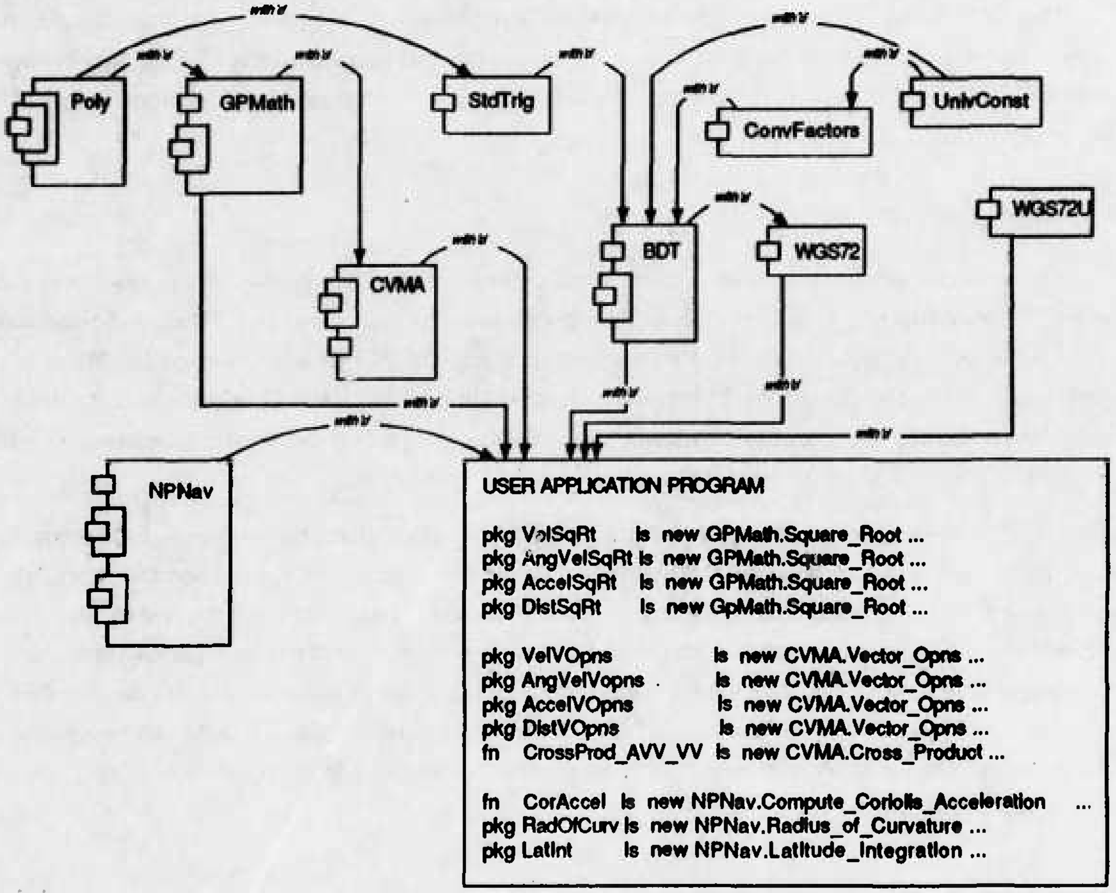
The size of the object code generated from the benchmarks reflects the qualities of the compiler, the CAMP parts, and, to a lesser extent, the instruction set architecture of the application target machine. Although RTE systems are being built with more and more memory, hardware capacity and its associated costs are still the major limiting factor in increasing the computational power of embedded applications. The execution benchmarks of the Benchmark Suite should facilitate the evaluation of Ada compiler/linker systems based on object code size. Linker map data, obtained by compiling and linking the benchmarks, can be utilized in judging an Ada system's appropriateness to an embedded application in the light of hardware capacity constraints.

2. COMPILATION BENCHMARKS

The purpose of the compilation benchmarks is to determine the compilability and linkability of a large selection of CAMP parts integrated into typical armonics application groupings. Results from compiling this series of benchmarks reflect on the ability of Ada compilers to correctly process CAMP parts. Since these parts are both reusable and armonics application-oriented, the validity of the benchmarks extends strongly to these two areas.

a. Compilation Group 1

CAMP parts utilized as benchmarks in Compilation Group 1 represent those which might be needed in a north-pointing navigation implementation. The structure, components, and operating procedure of this compilation benchmark follow.



```
USER APPLICATION PROGRAM

pkg VelSqRt      Is  new GPMath.Square_Root ...
pkg AngVelSqRt Is  new GPMath.Square_Root ...
pkg AccelSqRt    Is  new GPMath.Square_Root ...
pkg DistSqRt      Is  new GpMath.Square_Root ...

pkg VelVOpns                Is  new CVMA.Vector_Opns ...
pkg AngVelVopns .           Is  new CVMA.Vector_Opns ...
pkg AccelVOpns              Is  new CVMA.Vector_Opns ...
pkg DistVOpns               Is  new CVMA.Vector_Opns ...
fn    CrossProd_AVV_VV  Is  new CVMA.Cross_Product ...

fn    CorAccel  Is  new NPNav.Compute_Coriolis_Acceleration    ...
pkg RadOfCurv Is  new NPNav.Radius_of_Curvature ...
pkg LatInt        Is  new NPNav.Latitude_Integration ...
```

Figure 1. Compilation 1 Structure

- Compilation structure: Figure 1 depicts the compilation structure. An Ada main procedure is compiled in the context of several CAMP packages. The order of compilation for the packages corresponds to the partial ordering induced by the context clauses (*with* statements) of the packages and driver procedure. A command file in the tool set supplied with the benchmark suite gives a correct compilation order and compiles the compilation benchmarks automatically on VAX/VMS.

- Benchmark driver design:

    1. Import North_Pointing_Navigation_Parts (CAMP part number P001), General_Purpose_Math (P687), Coordinate_Vector_Matrix_Algebra (P681), Basic_Data_Types (P621), WGS72_Ellipsoid_Metric_Data (P611), WGS72_Ellipsoid_Unitless_Data (P613), and SYSTEM.
    2. Begin main procedure definition.
    3. Declare types and subtypes necessary for benchmark.
    4. Instantiate generic units from imported packages.
    5. Declare objects necessary for benchmark.
    6. Invoke instantiated and derived subprograms (executable part of driver).
    7. End main procedure definition.

- Data to be recorded:

    1. Successful compilation;
    2. Successful link;
    3. Object code size (size of load module produced, if any);
    4. CPU time consumed by the compiler.

- Methods for recording data: The source files for this compilation benchmark are compiled in one group with the source files for the others. Error-free compilation is indicated by the compiler through listings or by some other mechanism. CPU time consumption is noted when it is reported by the compiler. The driver program is then linked and the size of the executable image recorded.

b. Compilation Group 2

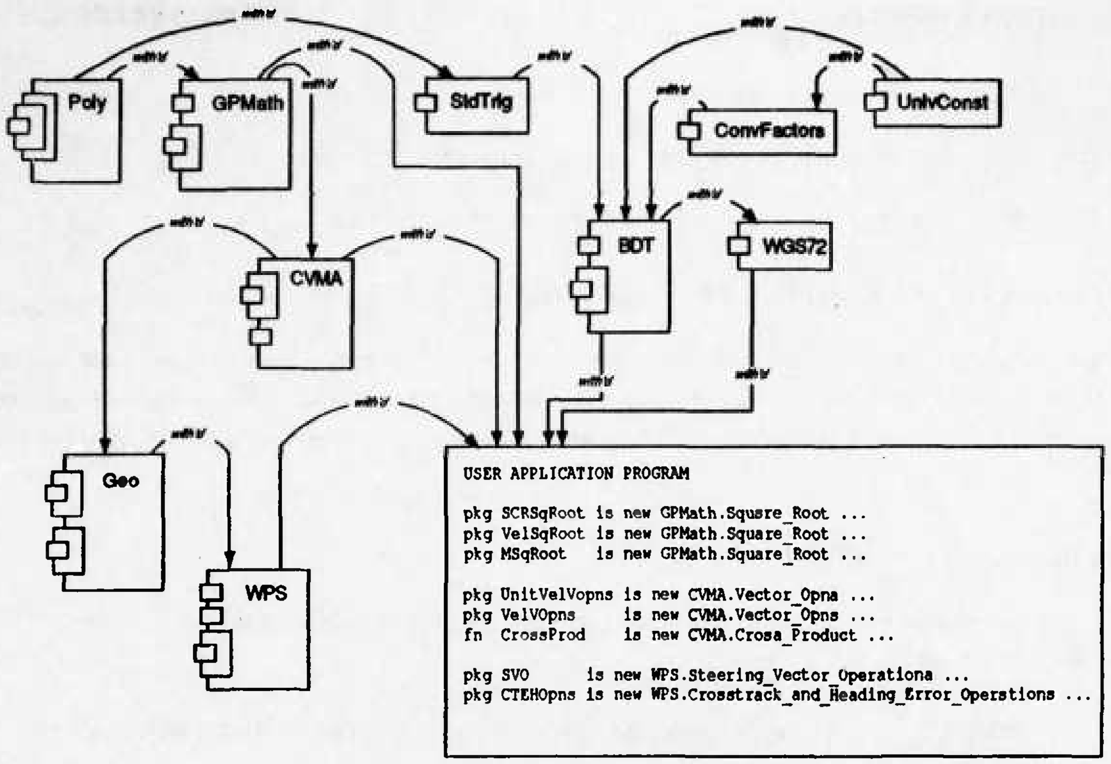
CAMP parts utilized as benchmarks in Compilation Group 2 represent those which might be needed in the waypoint steering of a missile application. The structure, components, and operating procedure of this compilation benchmark follow.

- Compilation structure: Figure 2 depicts the compilation structure. The structure is similar to that of Compilation Group 1.
- Benchmark driver design:
  1. Import Waypoint_Steering (CAMP part number P661), General_Purpose_Math (P687), Coordinate_Vector_Matrix_Algebra (P681), Basic_Data_Types (P621), WGS72_Ellipsoid_Metric_Data (P611).
  2. Begin main procedure definition.
  3. Declare types and subtypes necessary for benchmark.
  4. Instantiate generic units from imported packages.
  5. Declare objects necessary for benchmark.
  6. Invoke instantiated and derived subprograms.
  7. End main procedure definition.
- Data to be recorded: As in compilation group 1
- Methods for recording data: As in compilation group 1

c. Compilation Group 3

CAMP parts utilized as benchmarks in Compilation Group 3 represent those which might be needed in a Kalman filter of a missile application. The structure, components, and operating procedure of this compilation benchmark follow.

- Compilation structure: Figure 3 depicts the compilation structure. The structure is similar to that of the other two compilation groups.
- Benchmark driver design:
  1. Import Kalman_Filter_Complicated_H (CAMP part number P653) and Kalman_Filter_Data_Types (P622).
  2. Begin main procedure definition.
  3. Declare types and subtypes necessary for benchmark.
  4. Instantiate generic units from imported packages.



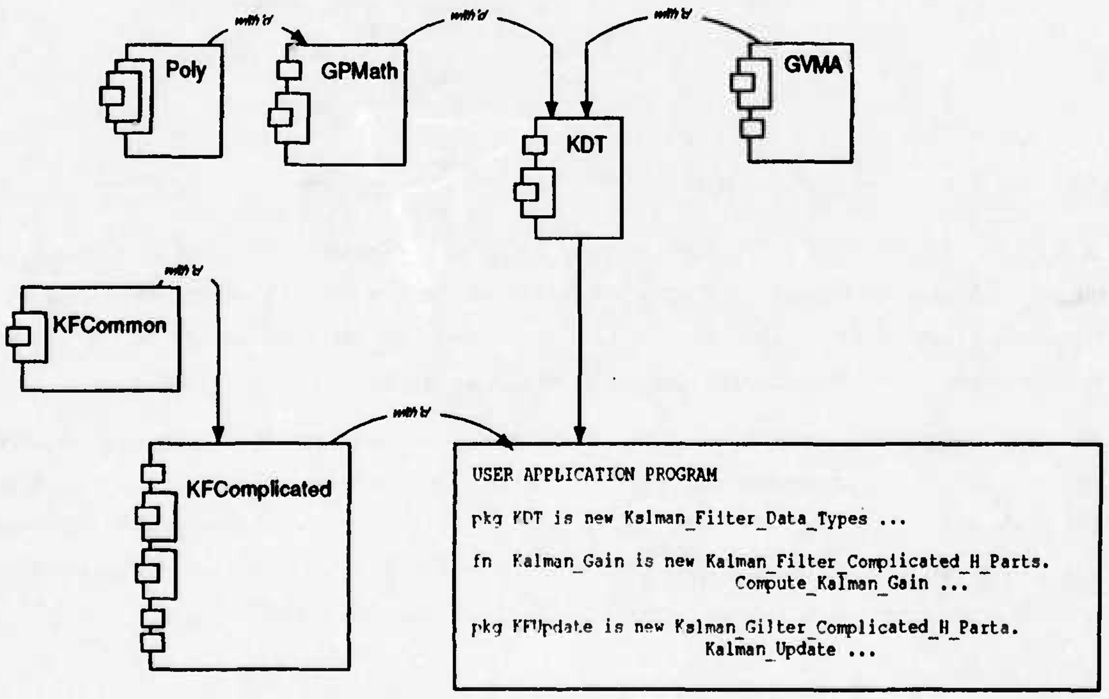
```
USER APPLICATION PROGRAM

pkg SCRSqRoot is new GPMath.Squsre_Root ...
pkg VelSqRoot is new GPMath.Square_Root ...
pkg MSqRoot   is new GPMath.Square_Root ...

pkg UnitVelVopns is new CVMA.Vector_Opna ...
pkg VelVOpns     is new CVMA.Vector_Opns ...
fn  CrossProd    is new CVMA.Crosa_Product ...

pkg SVO      is new WPS.Steering_Vector_Operationa ...
pkg CTEHOpns is new WPS.Crosstrack_and_Heading_Error_Operstions ...
```

Figure 2. Compilation 2 Structure



```
USER APPLICATION PROGRAM

pkg KDT is new Kalman_Filter_Data_Types ...

fn  Kalman_Gain is new Kalman_Filter_Complicated_H_Parts.
                         Compute_Kalman_Gain ...

pkg KFUpdate is new Kalman_Gilter_Complicated_H_Parta.
                      Kalman_Update ...
```

Figure 3. Compilation 3 Structure

5. Invoke instantiated subprograms.

6. End main procedure definition.

- Data to be recorded: As in the other two compilation groups

- Methods for recording data: As in the other two compilation groups

3. INTEGRATED EXECUTION BENCHMARKS

This section describes execution benchmarks based on CAMP parts, both integrated for use in a typical missile application and in a unit-testing environment. The purpose of the integrated execution benchmarks is to generate data on these CAMP parts and to afford an opportunity for determining code sizes.

a. Integrated Execution 1

In this section a benchmark based on a guidance computer implementation is described. Table 2 lists the CAMP parts used in this benchmark.

TABLE 2. CAMP PART BENCHMARKS OF INTEGRATED EXECUTION 1

TLCSC NAME	PART NO.	LLCSC NAMES
Waypoint Steering	P661	Compute Turn Angle and Direction Compute Turning and Nonturning Distances Distance to Current Waypoint Steering Vector Operations with Arcsine Turn Test Operations Cross Track and Heading Error Operations
Signal Processing Parts	P686	Absolute Limiter Upper Lower Limiter

- Benchmark Driver Design: This benchmark is based on the guidance computer of a missile application. The driver consists of several task bodies declared in the declaration section of a main procedure. These tasks are activated after the elaboration of the driver declaration section. A null executable part of the driver runs to completion and awaits the termination of the tasks.

  The tasks call the benchmark subprograms in the course of execution. A counter keeps track of calls to a central message management task. When the counter value reaches a certain level, the task is aborted and becomes abnormal. As the other tasks attempt to rendezvous with the aborted task, they are forced to select a "terminate" entry. Then, these tasks also become abnormal. When the child tasks of the driver have all become abnormal, the driver terminates execution.

- Data to be recorded:

  - Execution time

  - Code sizes

- Result data

- Methods for recording data: Execution time is obtained directly from the benchmark driver. The code sizes of the various CAMP parts may be taken from linker map files. Result data is also generated directly by the benchmark.

- Benchmark inputs: Before execution, the benchmark driver requests data about the system: the compiler used, the compiler host, and the compiler target. Then iteration values are requested to tell the driver how many times to execute a benchmark subprogram. The benchmarks themselves are supplied with hard-coded input data by the driver software. These inputs are coded as variables to preserve the functionality of the benchmarks, which would not normally process static data.

- Benchmark correct outputs: A file containing standard output is supplied with the benchmark suite. It should be used for comparison with the actual benchmark output.

b. Integrated Execution 2

In this section a benchmark based on a navigation computer implementation is described. Table 3 lists the CAMP parts used as benchmarks.

TABLE 3. CAMP PART BENCHMARKS OF INTEGRATED EXECUTION 2

TLCSC NAME	PART NO.	LLCSC NAMES
Common Navigation Parts	P001	Update Velocity Compute Ground Velocity Compute Gravitational Acceleration Sin Lat In
Wander Azimuth Navigation Parts	P002	Radius of Curvature Compute East Velocity Compute North Velocity Compute Latitude using 2-Value Arctan Compute Longitude using 2-Value Arctan Compute Wander Azimuth Angle Earth Rotation Rate Earth Relative Navigation Rotation Rates Compute Coriolis Acceleration Total Platform Rotation Rate
Direction Cosine Matrix	P644	CNE Operations
General Vector Matrix Algebra	P682	Matrix Matrix Multiply Restricted
General Purpose Math Parts	P687	Accumulator

- Benchmark Driver Design: This set of three benchmark drivers is based on the navigation operations of a missile application. Each driver uses the same basic Ada linkage closure of units, substituting dummy code as appropriate. The first phase of the benchmarking run is done by the driver, "Execute_Navigator_Test," which calls most of the benchmark subprograms. The remaining benchmark subprograms are called by two drivers embedded in the executable part of the Navigation Operations package.

- Data to be recorded:

1. Execution time

2. Code size

3. Result data

- Methods for recording data: As in Integrated Execution 1
- Benchmark inputs: As in Integrated Execution 1
- Benchmark correct outputs: As in Integrated Execution 1

c. Integrated Execution 3

In this subsection a benchmark based on the CAMP Kalman filter unit tests is described. Table 4 lists the CAMP parts used as benchmarks.

TABLE 4. CAMP PART BENCHMARKS OF INTEGRATED EXECUTION 3

TLCSC NAME	PART NO.	LLCSC NAMES
Kalman Filter Common Parts	P651	Error Covariance Matrix Manager State Transition and Process Noise Matrices Manager State Transition Matrix Manager
Kalman Filter Compact H Parts	P652	Compute Kalman Gains Update Error Covariance Matrix Update State Vector Sequentially Update Covariance Matrix And State Vector Kalman Update Update Error Covariance Matrix General Form
Kalman Filter Complicated H Parts	P653	Compute Kalman Gain Update Error Covariance Matrix Update State Vector Sequentially Update Covariance Matrix And State Vector Kalman Update Update Error Covariance Matrix General Form

- Benchmark driver design: The three drivers of this benchmark are based on the unit tests of the CAMP Kalman filter parts (P651, P652, and P653). Three main procedures import the three Kalman TLCSCs and call the benchmark subprograms within them.
- Data to be recorded:
  - Execution time
  - Code size
  - Result data
- Methods for recording data: As in the other two integrated execution benchmarks
- Benchmark inputs: As in the other two integrated execution benchmarks
- Benchmark correct outputs: As in the other two integrated execution benchmarks

4. POLYNOMIAL BENCHMARKS

The purpose of the polynomial benchmarks is to generate run-time data on the "slide rule" functions of the CAMP Polynomials package and to provide an opportunity for collecting object code size data. The run-time data on the benchmarks is produced by benchmark drivers and includes information both on the time-consumption of the benchmarks and the numeric output they produce.

Table 5 presents a summary of the execution benchmarks which have been created from the CAMP Polynomials parts. Entries marked "X" indicate a function and a numerical algorithm. For each mathematical function of the CAMP Polynomials package, all of the available algorithm implementations are used as benchmarks. The floating point types of the arguments and results are varied according to the number of terms in each algorithm's polynomial expansion. For example, an algorithm for a 5-term polynomial expansion may be instantiated to use 6 floating-point digits while an algorithm for a 7-term expansion is instantiated to use 9 digits.

TABLE 5. CAMP POLYNOMIAL PARTS EXECUTION BENCHMARKS

Function	Taylor Series	Modified Taylor Series	Hastings	Chebyshev	System Functions	Hart	Fike	Continued Fraction	Newton-Raphson	Modified Newton-Raphson	Cody-Waite
Sine	X	X	X	X	X						
Cosine	X	X	X		X	X					
Tangent	X	X	X		X			X			
Arcsine	X				X		X				
Arccosine	X				X		X				
Arctangent	X		X		X			X			
Square root									X	X	
Log 10					X						X
Log 2					X						X
Nat. Log											X

Tables 6 through 12 present details of the polynomial function benchmarking. An "X" entry in a table indicates an algorithm for computing a function and the number of terms of that algorithm to be applied in the computation. Detail tables are not included for the various log functions shown in Table 5 since the log function testing is confined to the Polynomials Cody-Waite LLCSC. Term counts are not applicable to the parts of this LLCSC.

- Benchmark driver design (for all polynomial benchmarks):

  1. Import the CAMP Polynomials package (P688), Benchmarking_Tools package, and the Polynomial_Benchmark package.

  2. Define a floating point type of some precision.

  3. Instantiate a Polynomials package LLCSC for the defined type.

  4. Instantiate the Polynomial_Benchmark package for the defined type.

TABLE 6. DETAIL OF SINE PERFORMANCE BENCHMARKS

Number of Terms	Taylor Series	Modified Taylor Series	Hastings Algorithm	Chebyshev Polynomial	System Functions
4	X	X	X		
5	X	X	X	X	
6	X	X			
7	X	X			
8	X	X			
VAX					X

5. Instantiate the Benchmark procedure (procedure named Benchmark) from the Polynomial_Benchmark package. Pass in a function subprogram as a generic actual parameter. This is the subprogram to be benchmarked. Pass in an identity function from the Benchmarking_Tools package as another generic actual parameter. This subprogram helps to compensate for time costs associated with the design of the benchmark driver software. A new Benchmark procedure instantiation is required for each subprogram benchmark from the Polynomials package.

6. Request the system information. This includes the name of the compiler used to compile the benchmark and the names of the host and target machines of the compiler. This data is incorporated into the benchmark driver output to note the environment in which the benchmark is being carried out. See "Benchmark Correct Outputs" below.

7. Request the number of iterations to use for each benchmark. Separate numbers are requested: one for the number of iterations to use when timing the benchmark, the other for the iterations to use when collecting data from the benchmark.

8. Call the instantiated Benchmark procedures. These procedures time the benchmark subprogram over a selected domain. They also provide input and output data echoing for the benchmark subprogram over the argument domain.

9. End of benchmark definition.

- Benchmark inputs: System information and iteration values are supplied at run-time via the console.

- Benchmark correct outputs: The benchmarks produce time-consumption data as well as: echoed system information (noted above), an Ada enumeration literal for the function being benchmarked (e.g. SINE_R for radian sine), and ordered pairs of benchmark subprogram input and output. Accuracy of the subprogram output is determined by an analysis program supplied with the Benchmark Suite. This program uses the VAX Ada Math library (MATH_LIB) to obtain truth values. Absolute error in a benchmark subprogram is calculated as the difference between the result of that subprogram and the truth value result for a given argument.[2]

- Data to be recorded:

    1. Execution time for one call to each Ada subprogram benchmark

    2. Code size

    3. Arguments and benchmark function results for those arguments

    4. System information collected at the beginning of the run

- Methods for recording data: Time-consumption data is recorded and reported automatically by the benchmark drivers. Input data, output data, system information echoing, and an enumeration literal representing the kind of function benchmarked are also reported automatically. Analyzed output is obtained by passing the benchmark driver output through the analysis program Analyze. Code size information is retrieved from linker maps.

TABLE 7. DETAIL OF COSINE PERFORMANCE BENCHMARKS

Number of Terms	Taylor Series	Modified Taylor Series	Hastings Algorithm	Hart Algorithm	System Functions
4	X	X	X		
5	X	X	X	X	
6	X	X			
7	X	X			
8	X	X			
VAX					X

[2]Note: a small amount of error is induced by conversion to and from text representations of floating-point numbers.

TABLE 8. DETAIL OF TANGENT PERFORMANCE BENCHMARKS

Number of Terms	Taylor Series	Modified Taylor Series	Hastings Algorithm	Continued Fraction	System Functions
4		X	X	X	
5		X	X	X	
6		X		X	
7		X		X	
8	X	X		X	
9				X	
10				X	
11				X	
VAX					X

TABLE 9. DETAIL OF ARCSINE PERFORMANCE BENCHMARKS

Number of Terms	Taylor Series	Fike Semicircle	System Functions
5	X		
6	X	X	
7	X		
8	X		
VAX			X

TABLE 10. DETAIL OF ARCCOSINE PERFORMANCE BENCHMARKS

Number of Terms	Taylor Series	Fike Semicircle	System Functions
5	X		
6	X	X	
7	X		
8	X		
VAX			X

TABLE 11. DETAIL OF ARCTANGENT PERFORMANCE BENCHMARKS

Number of Terms	Taylor Series	Alternate Taylor Series	Hastings Algorithm	Modified Hastings Algorithm	Continued Fraction	System Functions
4	X	X			X	
5	X	X			X	
6	X	X	X	X	X	
7	X	X	X	X	X	
8	X	X	X	X	X	
9					X	
10					X	
11					X	
VAX						X

TABLE 12. DETAIL OF SQUARE ROOT PERFORMANCE BENCHMARKING

Number of Terms	Newton-Raphson	Modified Newton-Raphson	System Functions
4			
5			
6	X	X	
7			
VAX			X

SECTION IV

METHODOLOGY

The following paragraphs explain methods used in constructing the Armonics Benchmark Suite. These sections discuss the overall design aspects of the suite as applied to the problems of portability, validity, and automation of data collection.

1. PORTABILITY

Like the CAMP parts in general, the Benchmark Suite is highly portable, extending its usability and repeatability to many different Ada systems. The CAMP parts selected as benchmarks use only Mil-Std-1815A Ada code, as do the drivers which automate much of the benchmarking. Whenever optional Ada features are applied (e.g., pragma PAGE), their effects are irrelevant and they may be freely ignored by Ada compilers.

Input to and output from the benchmarks is limited to the use of the console, obviating file I/O implementation in the target system. While a filing system is desirable in order to retain output, the console I/O approach possesses greater versatility since many embedded computers (and hosted debugger/simulators for the same) may not fully support file I/O. In such cases, the use of file I/O could make the benchmarks difficult to transport to the kinds of architectures for which they are intended. Moreover, the use of console I/O does little to impede the retention of benchmark data on a filing system. The Ada language and most operating systems supply trivial mechanisms for redirecting console output to files.

The tool set which accompanies the benchmark suite is system-dependent and, as previously noted, consists of VAX DCL command procedures and some non-portable Ada. Designed to automate the compilation and execution of the benchmarks, this tool set supports two possible uses: For VAX/VMS users, the tool set substantially automates benchmarking; for users of other systems, the tools are well documented to permit a knowledgeable user to modify them or use them as a guide for performing the benchmarks on his own system. A more detailed treatment of the tools is presented in Section V.

In order to automate the timing of benchmark executions in a portable way, the benchmark drivers use facilities from the Ada CALENDAR package. Although differences in the implementation of this package may exist between systems, these differences are minor enough that their effects can be minimized. The design of the benchmark drivers attempts to take advantage of similarities in Ada systems supporting the CALENDAR package, while accounting for the differences that exist.

For example, the duration of a basic clock cycle (Ada SYSTEM.TICK) may vary from system to system and may be quite large with respect to the benchmark execution times. This requires a benchmark driver to execute its benchmark many times in order to arrive at a reasonable one-call execution-time estimate. By examining the Ada constant SYSTEM.TICK, the drivers are able to calculate the number of benchmark executions necessary to arrive at a set timing accuracy. Conversely, given the number of executions used in benchmarking, the drivers calculate an estimated accuracy on the time-consumption data obtained from those executions.

2. COMPILATION BENCHMARK METHODS

The compilation benchmarks are valid tests of a compiler but do not apply to code generation. They are intended to force an Ada compiler/linker system to fail when it contains errors in semantic analysis, in library management, or in linkage editing.

To pass the compilation benchmark test, a compiler must process the associated Ada source code without signaling any errors or ending abnormally. Limited warnings are allowed since the Ada language allows a compiler some flexibility. For example, a compiler can warn that it has made an optimization or ignored an optional pragma. Warnings about program semantics, however, should not be generated, nor should the compiler or linker encounter fatal errors in library management or load module generation.

The compilation benchmarks are performed by a DCL command procedure. This procedure must be supplied with a compiler invocation command and a linker invocation command. It then proceeds to apply these commands to the necessary Ada source files in a correct order. The procedure can be altered or used as a guide when benchmarking on systems other than VAX/VMS.

The compilation benchmarks were validated by successfully compiling, linking, and running their source code on a highly reliable Ada compiler/linker system. The system on which the validation was performed is ACVC validated and produced no error messages or warnings in the course of compiling, linking, and running the compilation benchmark source code.

3. EXECUTION BENCHMARK METHODS

a. Collecting Valid Timing Data

Drivers of the execution benchmarks collect time-consumption data through the use of the Ada CALENDAR package. As noted above, the Ada constant SYSTEM.TICK varies between systems and is usually quite large. Because of this the benchmarks are called repeatedly for the sake of timing accuracy. The number of repetitions necessary to achieve microsecond accuracy is computed relative to SYSTEM.TICK and reported by the benchmark drivers at run-time. This enables a user of the Benchmark Suite to decide on the number of repetitions to actually use in benchmarking.

The computed number of repetitions is not used automatically since the resulting processing time of the benchmark drivers might, in some cases, become prohibitively long. Large numbers of repetitions on slow systems may consume a substantial amount of CPU time. Reducing the number of repetitions proportionally reduces both the driver CPU-time expense and, unfortunately, the accuracy of the collected benchmark timings. The engineer who runs the benchmarks must, therefore, make a trade-off with respect to timing accuracy and the use (or overuse) of computational resources.

In order to ensure valid timings of the benchmarks, a number of precautions are built into the system. Code optimizations which might affect the integrity of the time-consumption data are selectively defeated while other optimizations remain untouched. The methods used are similar to those used in ACVC tests to prevent the compile time removal of code which is being tested. These methods entail the use of identity functions and tautological BOOLEAN functions to deprive the compiler of optimization opportunities. Here, the goal of optimization suppression is essentially to "fool" the compiler by reducing its available control-flow information at compile time.

For example, consider Figure 4. The identity function used in the first part of this Ada fragment prevents a compiler from propagating the constant "5" into the timing loop in place of the variable Argument. If this propagation were allowed to occur, the measured time for the subprogram call could be reduced. The reduction, however would be due to the static nature of the argument, a circumstance which would not often occur if the subprogram were used in an application. This method of optimization suppression is used in the integrated execution benchmarks where constants are often supplied as arguments.

```
          ...

Argument := Identity(5);  -- instead of "Argument := 5;"
Start_Timer;
for Index in Some_Range loop
   Result := Benchmark_Function (Argument);
end loop;
Stop_Timer;

          ...
```

Figure 4. Identity Function Defeats Constant Propagation

In a second example, Figure 5 shows the use of a tautological BOOLEAN function to prevent the removal of a "dead" assignment. The function, Snow_Is_White, always returns the value TRUE, although this is not known at compile time by the compiler (the body of the function is separate). Since the flow of control is not known, the compiler cannot remove the assignment to the variable Gross_Time. If this optimization were allowed, the compiler could move the evaluation of the Get_Elapsed_Time_Since_Start function into the expression assigned to the variable Next_Time_Used. While such a move would not alter the logical meaning of the program, a small effect on the time data would result. This optimization suppression is used in the polynomial benchmarks.

It should be noted that these techniques do not have the negative effect of inhibiting desired optimizations. An Ada compiler is free to optimize all unprotected source code, including the code bodies of the benchmarks themselves. It should also be noted, however, that these techniques are not fool-proof. Conceivably, a smart enough compiler/linker could outwit the optimization suppressions described here.

```
        ...

Start_Timer;
for Index in Some_Range loop
   Argument := Identity (Argument);   -- loop determines overhead
end loop;
if Snow_Is_White then                           -- balances "if" below.
   Overhead_Time := Get_Elapsed_Time_Since_Start;
end if;

Start_Timer;
for Index in Some_Range loop
   Argument := Identity (Argument);
   Result   := Benchmark_Function (Argument);
end loop;
if Snow_Is_White then                           -- function always TRUE
   Gross_Time := Get_Elapsed_Time_Since_Start;  -- removal prevented
end if;
Net_Time_Used := Gross_Time - Overhead_Time;

        ...
```

Figure 5. Tautological Function Prevents Assignment Removal

Figure 5 also illustrates the collection of time-overhead to calibrate benchmark timings. Time overheads are calculated at run-time and are used to offset the effects of the timing method and benchmark code idiosyncracies. In the case of the polynomial benchmarks, the overheads are often negligible, since none of these benchmarks require initialization.

On the other hand, the execution time overheads of the integrated execution benchmarks are usually appreciable due to the parameter requirements of the benchmark subprograms. Many of the higher-order CAMP parts used as integrated execution benchmarks have side effects and in-out parameters. Each execution of such a benchmark, therefore, has a cumulative effect which may produce an exception after many iterations. In order to counteract this effect, many of the integrated execution benchmarks must be re-initialized prior to each call, a process which adds very significantly to overhead. The resolution of this problem is transparent to the user and is accomplished by, as in the polynomial benchmarks, implementing automatic overhead correction in the benchmark drivers.

Despite all of the precautions taken to ensure the validity of the time-consumption data, inaccuracies may still occur. The benchmark drivers may overestimate benchmark execution times when asynchronous events take place in the midst of timing. For example, if the benchmark drivers are operated on a time-sharing operating system, they will compete with other processes. Since the CALENDAR package operates on wall-clock time rather than CPU time consumption, the benchmarks will appear to execute longer as their CPU time fraction is reduced.

Problems of this kind are beyond the control of the benchmark drivers. The effects of asynchronous events on the benchmark timings may be minimized, but inaccuracies should nevertheless be assumed. When asynchronous interference with the benchmarks is relatively uniform, the benchmark execution times will lengthen proportionally to their synchronous execution times. Benchmarks which, by themselves, take a relatively long time to run will, of course, show a relatively larger dilation in measured execution time. While this effect may be undesirable, it can usually be taken into account.

Moreover, timings which include asynchronous interference, typical of an operating environment, are quite valid. In such an environment, estimates based on the CPU time consumption alone would be unrealistically low. The true throughput of an application is a function of both the application execution speed and the typical amount of asynchronous interference with which the application must contend.

b. Collecting Benchmark Output Data

In addition to timing data, the polynomial benchmarks provide data for use in determining the accuracy of Polynomials (CAMP package) function results. For each function benchmarked, both input and output are reported at equal intervals over a selected argument domain. This permits the result accuracies of the functions to be checked against appropriate truth values. For users with access to VAX Ada, accuracy analysis and report generation can be accomplished automatically using a tool provided with the Benchmark Suite. Other users may make use of this tool by modifying it as explained in section V. It should be noted, however, that the accuracy analysis tool is not required in order to run the benchmarks.

Armonics subsystem output data is also produced by the integrated execution benchmarks although automatic checking of this data is not supported. Most of the output from these benchmarks is produced in an ad hoc format which does not lend itself to automatic analysis. Nevertheless, the correctness of the data may be checked by manual comparison with standard output files supplied with the Benchmark Suite.

c. Automation of the Execution Benchmarks

The compilation and implementation of the execution benchmarks is highly automated in the Armonics Benchmark Suite. Depending on the computer system used, most of the benchmarking, from installation to report generation, may be accomplished in one or two man-days. In addition, once an engineer has conformed the Benchmark Suite to run on a particular system, the work can be easily repeated as necessary.

Compilation of the source code of the execution benchmarks is explained in detail in the next section. The process involves the use of VAX/VMS command procedures as discussed above for the compilation benchmarks. Once again, these command procedures may be used directly on a VAX, modified on systems which support batch processing, or used as a guide on other systems.

The process of *running* the benchmarks is automated at two levels. First, the benchmarks themselves (i.e. the chosen CAMP parts) are automatically executed by the portable Ada drivers in which they are embedded. Thus, the engineer with the task of benchmarking is not required to supply an Ada driver with which to execute the benchmarks. Second, at a higher level, the benchmark drivers are executed using VAX DCL command procedures, written for inclusion with the Benchmark Suite. This level of automation is, of course, subject to system dependencies.

SECTION V

USING THE BENCHMARKS

The following sections explain how to perform benchmarking using the CAMP Armonics Benchmark Suite. For the purposes of discussion, the VAX/VMS environment is assumed. Comments throughout suggest possible ways of adapting the Benchmark Suite to other environments.

Table 13 lists the DCL command procedure files which are supplied with the Benchmark Suite. On a VAX, these command procedures automate both the compilation and the execution of the benchmark drivers. On other systems the command procedures serve as a guide although they may be altered as necessary to conform to other batch-processing systems.

1. LOGICAL DIRECTORIES

A system of three directories is recommended for compiling and executing the benchmark code in a VAX/VMS environment. These directories are referred to within the Benchmark Suite command procedures by the following VMS logical names:

- Compilation_Directory: The directory on which all compilation takes place.
- Tools: The directory which contains the Benchmark Suite command procedures, and
- Source: The directory which contains all of the Ada source code supplied with the benchmarks.

On systems which do not support the concept of logical names, the command procedures may be altered to use the desired operating system names and batch job control style. Systems which do not support the concept of directories at all may store all of the files (over 360 in number) in a single location and alter the command procedures accordingly.

2. USING THE COMPILATION BENCHMARKS

The compilation benchmarks are simply files of Ada source code. Testing a compiler/linker system with the benchmarks involves compiling the Ada code in a correct order and then linking the three linkable main procedures. For VAX/VMS-hosted Ada compilers the process is automatic depending slightly on the command syntax used to invoke the subject Ada compiler and linker.

The file called VAX_Compilation_Run.Com gives an example of how to perform the compilation and linkage editing of the compilation benchmarks on VAX/VMS, using the VAX Ada compiler and (via ACS) the VMS linker. The procedure sets its process to run in the logical Compilation_Directory and then calls the command procedure Compilation_Benchmarks to perform the compilation and the linkage editing. On other systems, the Compilation_Benchmarks procedure gives a correct compilation order for the Ada source files and may be used as a guide or altered as necessary.

TABLE 13. BENCHMARK SUITE COMMAND PROCEDURES

COMMAND PROCEDURE	PURPOSE
ACT_COMPILATION_RUN	Calls Compilation_Benchmarks to compile/link the compilation benchmarks on the ACT compiler.
ANSI2DV & DV2ANSI	Rename files from ANSI to development names and reverse.
COMPILATION_BENCHMARKS	Compiles/links source code for the compilation benchmarks.
COMPILE_BENCHMARK_SUPPORT	Compiles support code for the execution benchmarks.
COMPILE_TOOLS	Compiles the clock function and I/O tools of the execution benchmarks.
INT_EXEC1_COM_LINK	Compiles/links Ada source code for integrated execution 1
INT_EXEC2_COM_LINK	Compiles/links Ada source code for integrated execution 2
INT_EXEC3_COM_LINK	Compiles/links Ada source code for integrated execution 3
MODIFIED_POLY6_COM_LINK	Compiles/links the 6-digit precision polynomial benchmarks on the TLD compiler.
MODIFIED_POLY9_COM_LINK	Compiles/links the 9-digit precision polynomial benchmarks on the TLD compiler.
POLY6_COM_LINK	Compiles/links the 6-digit precision polynomial benchmarks.
POLY9_COM_LINK	Compiles/links the 9-digit precision polynomial benchmarks.
SYSTEM_COM_LINK	Compiles/links code to run the System polynomial benchmark.
TLD_BENCHMARKS_COM_LINK	Calls other procedures to compile/link the benchmarks on the TLD compiler.
TLD_COMPILATION_RUN	Calls Compilation_Benchmarks to compile/link the compilation benchmarks on the TLD compiler
VAX_ANALYZE_COM_LINK	Compiles/links the Analyze.Ada program. The program is VAX/VMS and VAX Ada dependent.
VAX_ANALYZE_POLY	Uses Analyze.Ada to analyze all of the output from the polynomial benchmarks.
VAX_BENCHMARKS_COM_LINK	Calls other procedures to compile/link the benchmarks on the VAX Ada compiler.
VAX_COMPILATION_RUN	Calls Compilation_Benchmarks to compile/link the compilation benchmarks on the VAX Ada compiler.
VAX_INT_EXEC1_RUN	Runs integrated execution 1 on the VAX.
VAX_INT_EXEC2_RUN	Runs integrated execution 2 on the VAX.
VAX_INT_EXEC3_RUN	Runs integrated execution 3 on the VAX.
VAX_POLY_RUN	Runs the polynomial benchmarks on the VAX.

3. COMPILING THE POLYNOMIAL AND INTEGRATED EXECUTION BENCHMARKS

The two classes of executable benchmarks in the Armonics Benchmark Suite must be compiled and linked prior to benchmarking. On VAX/VMS, this process is automatic and is accomplished by the VAX_Benchmarks_Com_Link command procedure provided with the Benchmark Suite. This command procedure establishes a process in the logical Compilation_Directory and then proceeds to call other Benchmark Suite command procedures to accomplish the various compilation and linkage editing tasks. The following command procedures are performed in order:

1. Compile_Benchmark_Support: compiles the CAMP and 11th Missile software used in benchmarking. This software contains the actual benchmarks (i.e., the CAMP parts selected as benchmarks) as well as necessary support code. After compilation, this software comprises a library of Ada units which provide a context for the subsequent compilation of the benchmark drivers.

2. Compile_Tools: compiles packages of benchmarking tools used by the drivers. These packages are fully portable and provide the drivers with necessary I/O routines and other utilities.

3. VAX_Analyze_Com_Link: compiles and links the tool, Analyze, used to analyze the output of the polynomial benchmarks. This tool is dependent on VMS and VAX Ada as explained in Section III. The Benchmark Suite program library dependency of this tool is limited to the package Benchmarking_Tools, compiled by the procedure, Compile_Tools, just discussed. This means that the analysis tool may be independently compiled on VAX/VMS and VAX Ada and then used to check the polynomial benchmark output from other systems.

4. Poly6_Com_Link and Poly9_Com_Link: compile and link the polynomial benchmark drivers. Two command procedures are used: one for the drivers using 6-digit Ada floating point numbers, and one for the drivers using 9-digit numbers. Thus, Ada systems which do not support the extended floating-point representations may still compile the lower-accuracy drivers without difficulty. It should be noted, however, that the Ada source code files of Poly9_Com_Link will not correctly compile unless those of Poly6_Com_Link have already been compiled. Two packages necessary to the polynomial benchmark drivers of both precisions are compiled in Poly6_Com_Link.

5. Int_Exec1_Com_Link, Int_Exec2_Com_Link, and Int_Exec3_Com_Link: compile and link the integrated execution benchmarks. Each of these command procedures compiles the support and drivers necessary to run the respective integrated execution benchmarks.

6. System_Com_Link: recompiles CAMP Polynomials support on the VAX then compiles and links the System Driver benchmarks. This driver uses the VAX Ada math library as a set of benchmarks. The Polynomials System_Functions LLCSC interfaces this driver to the math library. This can be of interest to users of VAX/VMS and VAX Ada who must use the VAX Ada "slide rule" functions, but who have to meet real-time constraints.

4. RUNNING THE EXECUTION BENCHMARKS

a. Polynomial Benchmark Execution

Once compiled and linked, the polynomial benchmark drivers may be run to produce data. As has been discussed, these drivers send output to standard output which is generally the console. On most systems supporting file I/O, including VMS, the standard output can be redirected to files.

The command procedure VAX_Poly_Run is an example of running the polynomial benchmarks in the VMS environment. Standard input is redefined to permit the drivers to request their Input from a file. This file, created automatically by VAX_Poly_Run at benchmark time, contains data to supply the benchmark drivers with the following:

- Compiler Name: the name of the compiler used to compile the polynomial drivers (becomes part of the output data).
- Host Name: the name of the compiler host machine (becomes part of the output data).
- Target Name: the name of the target machine of the compiler and the machine on which the benchmarks will run (becomes part of the output data).
- Number of timing iterations: the number of times that a driver must call a function in order to achieve a certain accuracy in calculating the time for a single call.
- Number of data iterations: the number of data values to use as arguments to a function of the driver. This defines the number of argument-result pairs produced as output for each benchmark of the driver.

Standard output, like standard input, is also redefined in the case of each benchmark driver to channel output to files. This permits the subsequent analysis of the output by the analysis program Analyze.

The analysis program is not portable from the VMS and VAX Ada environment due to use of the VAX Math_Lib. Thus, use of the program on other systems is prohibited unless modifications are made. The program may, however, be modified by interfacing it to another math library, as long as the output from new math library has greater than nine Ada digits of precision. This is necessary since the math library is used by the analysis program to check the results of the polynomial benchmarks, which use up to nine digits of accuracy.

Running the analysis program is trivial and is demonstrated by the VAX_Analyze_Poly command procedure. A user simply executes the analysis program, Analyze, and provides it with the name of a data file produced by running the polynomial benchmark drivers. The program prompts again to request the name of the file in which the analyzed output is to be placed. After the analysis of a file is complete the program starts over, requesting the name of the next input file. If no file name is provided, the program terminates.

It should be noted that, although the analysis program is non-portable, it may be used to analyze polynomial benchmark data from diverse systems. A user with access to VAX/VMS and VAX Ada may use the analysis program exclusively on that system to check the benchmark output from many other systems.

b. Integrated Execution Benchmarks

Running the integrated execution benchmarks is similar to running the polynomial benchmarks. The command procedures VAX_Int_Exec1_Run, VAX_Int_Exec2_Run, and VAX_Int_Exec3_Run automatically execute the three integrated execution benchmark groups on VMS. These procedures provide the input data required by the drivers while the output of the drivers is trapped in log files by VMS.

Like the polynomial benchmark drivers, the integrated execution benchmark drivers use only standard I/O. The input data required by each of the drivers is as follows:

- Compiler Name: the name of the compiler used to compile the polynomial drivers (becomes part of the output data).
- Host Name: the name of the compiler host machine (becomes part of the output data).
- Target Name: the name of the target machine of the compiler and the machine on which the benchmarks will run (becomes part of the output data).
- Numbers of Timing Iterations: a series of numbers telling the driver how many times to execute corresponding benchmarks. Unlike the polynomial benchmarks, the different integrated execution benchmarks within a driver do not all have to be executed the same number of times. Also, overhead timing iterations vary from benchmark to benchmark. The command procedures which run the integrated execution benchmarks on VAX/VMS may be consulted for more details.

The output generated by running the integrated execution benchmarks consists of two types of data:

1. Result data, which represents the results of the calculations performed by the subprograms chosen as benchmarks and,
2. A table of timing data showing the time used for a single call to each benchmark subprogram.

Output data of the first type is to be used in checking the *correctness* of the data processing of a tested system. Such output should closely match the corresponding standard data supplied with the Benchmark Suite. The second type of data represents the *run-time efficiency* of the tested system and is expected to vary widely from system to system.

Although both kinds of data are produced with each run of an integrated execution benchmark driver, the correctness of the two types is mutually exclusive in a given run. A run which provides accurate run-time efficiency data is, by design, likely to produce poor data for correctness checking. The reverse is also true. For this reason, each integrated execution benchmark must be run twice, once for timing purposes and once to obtain data for comparison to supplied standard data.

When performing the timing run, the number of iterations for each benchmark subprogram (specified by the user at run-time) must be high in order to compensate for the generally low accuracy of the clock functions. Each subprogram will then be called many times, the time for one call being calculated by simple division. To aid the user, each benchmark driver reports the number of iterations necessary to obtain microsecond accuracy. Also, whatever number the user specifies, the resultant table of timings will show estimates of the accuracy actually obtained.

On the other hand, performing the benchmark run for correctness of data processing requires that the benchmark subprograms be executed only once. Thus, the user must specify that only one iteration be used for each subprogram. More than one call to a given subprogram can alter the output data, making any comparison to the standard invalid. This is due to the use of in-out parameters and occasional side effects in the benchmark subprograms. Results, in these cases, tend to accumulate changes from call to call as previously discussed.

APPENDIX A

ARMONICS BENCHMARK SUITE

This appendix presents a summary of the data which CAMP obtained from the Armonics Benchmark Suite. In some cases the data represents performance parameters of the selected CAMP parts as they operate on a 32-bit minicomputer. However, when possible, data reflecting the operation of the benchmarks in a Mil-Std-1750A microprocessor environment has been included.

The compilation benchmark data underscores some of the difficulties a software engineer may experience when selecting or applying an Ada compiler. It was found that many validated Ada compilers currently lack the ability to handle complex source code. The problem is essentially one of relative reliability: Some Ada compilers seem to work all of the time; most Ada compilers seem to work some of the time.

The polynomial benchmarks, which measure run-time parameters of Polynomials scientific functions, were executed successfully in both the 32-bit minicomputer and 1750A microprocessor environments. This supplied us with data enabling us to draw some useful conclusions about the CAMP parts, the Ada language, and the tested compiler/processor pairs. Finally, performance data from the integrated execution benchmarks serves to validate these benchmarks. At the time of this writing these benchmarks could not be run in any but the 32-bit environment due to errors in compilation to the 1750A target machine.

1. COMPILATION BENCHMARK DATA

The compilation benchmarks were used to test four separate Ada Compiler/Linker systems. One compiler, Compiler A, was self-targeted and served, because of its demonstrated reliability, as the validation compiler for the benchmarks. The other three compilers, B, C, and D, were recently validated cross-compilers to a 1750A target.

Compiler A succeeded in compiling all of the source code of the compilation benchmarks. It produced no warnings and no errors. The accompanying linker subsequently produced load modules with no difficulties. As a final step, the load modules were run on the host system to see if they would produce run-time errors. On this host, no errors occurred although this implies no guarantees about other systems.

Compiler B succeeded in compiling all of the source code correctly except the driver of the Kalman filter compilation benchmark, Compilation 3. Numerous warnings were issued in the course of compilation. The vast majority of these warnings concerned optimizations which could have been made in the Ada code but, for reasons of readability, were not. The compiler had performed an optimization that was not made by the programmer at the source code level. The warnings produced by Compiler B were justified with the exception of two concerning program semantics.

In compiling the Kalman filter driver, Compiler B evidently lost track of a necessary file. Object code was still generated but it was probably erroneous. Nevertheless, all three drivers were successfully linked, albeit with one warning. The linker of Compiler B produced the required load modules and did not fail to note that the Kalman filter driver had compiled with errors. The load modules were next loaded into the MDAC-Huntington Beach Mil-Std-1750A Simulator and their sizes were recorded.

Compiler C compiled all of the support packages of the compilation benchmarks but failed to compile any of the three drivers. In all three cases, the compiler ended abnormally in a late phase of processing. For this reason, Compiler C's linker could not be fairly tested.

Compiler D had been validated very recently and appeared to be having many of the problems associated with any new compiler. It failed to compile even the support packages of the compilation benchmarks. After successfully compiling the first three Ada files, the compiler falsely diagnosed the fourth as having semantic errors. Continuing through the source code, the compiler found numerous other "errors" in the error-free code.

A summary of the data collected on Compilers A, B, and C is presented in Table A-1. Insufficient data was obtained from Compiler D to justify its inclusion in the table. It should be noted that the object code size data for Compiler A may be unrealistically small. The size mentioned in the table does not include any run-time system services which may be required.

TABLE A-1. COMPILATION BENCHMARK DATA

COMPILER/ LINKER	SUCCESSFUL COMPILE?	SUCCESSFUL LINK?	TOTAL CPU TIME (secs.)	TOTAL OBJECT CODE SIZE
A	Yes	Yes	10:56.56	62K bytes
B	Most	Yes	22:42.33	122K bytes
C	No	NA	30:00.00?	?

2. POLYNOMIAL BENCHMARK DATA

The polynomial benchmarks were used to test two subject systems. System A consisted of Compiler A, above, and the host/target system of that compiler. System B consisted of Compiler B and the MDAC Huntington Beach Mil-Std-1750A simulator, which simulates a 1750A bare machine. Running the benchmarks on System A produced performance data on the CAMP Polynomials package parts as they run on a 32-bit time-sharing minicomputer. System B produced data for the same parts as they run on a 20 MHz 1750A microprocessor. Compilers C and D, above, failed to compile the polynomial benchmarks.

For each function of the Polynomials package, size data was obtained on System B. It was felt that 1750A size data was relevant to armonics applications. Moreover, this data was readily available in the linkage map files produced by the linker of System B. On the other hand, size data on the 32-bit system was less meaningful and was excluded. System A makes extensive use of built-in service routines which are not counted in load size; on a bare machine, system services are part of the load module or run-time system and are counted — a fact which casts doubt on the validity of code size estimates. Table A-2 gives the size data for functions of the Polynomials package on System B.

Time-consumption and mathematical precision data on the polynomial functions was collected for both systems A and B. This data is summarized in Figures A-1 to A-12. Each graph plots the execution

TABLE A-2. SYSTEM B POLYNOMIALS SIZES

TLCSC Name / LLCSC Name / Unit Name	Size (words) Hex.	Dec.
Chebyshev		
Radian_Operations		
Sin_R_5Term	5D	93
Degree_Operations		
Sin_D_5Term	5D	93
Semicircle_Operations		
Sin_S_5Term	5A	90
Cody_Waite		
Natural_Log		
Nat_Log	59	89
Base_N		
Log_N	12	18
Continued_Fractions		
Radian_Operations		
Tan_R	31	49
Arctan_R	36	54
Fike		
Semicircle_Operations		
Arcsin_S_4Term	60	96
Arcsin_S_5Term	64	100
Arcsin_S_6Term	68	104
Arccos_S_4Term	62	98
Arccos_S_5Term	66	102
Arccos_S_6Term	6A	106
Hart		
Radian_Operations		
Cos_R_5Term	52	82
Degree_Operations		
Cos_D_5Term	51	81
Hastings		
Radian_Operations		
Sin_R_4Term	36	54
Sin_R_5Term	3A	58
Cos_R_4Term	3D	61
Cos_R_5Term	41	65
Tan_R_4Term	25	37
Tan_P_5Term	25	37
Arctan_P_6Term	26	38
Arctan_R_7Term	2A	42
Arctan_R_8Term	2E	46
Mod_Arctan_R_6Term	4C	76
Mod_Arctan_R_7Term	50	80
Mod_Arctan_R_8Term	54	84
Degree_Operations		
Sin_D_4Term	36	54
Sin_D_5Term	3A	58
Cos_D_4Term	3D	61
Cos_D_5Term	41	65
Tan_D_4Term	23	35
Tan_D_5Term	23	35
Mod_Newton_Raphson		
Sqrt	61	97
Newton_Raphson		
Sqrt	6C	108
Taylor Series		
Radian_Operations		
Sin_R_4Term	34	52
Sin_R_5Term	37	55
Sin_R_6Term	3A	58
Sin_R_7Term	3D	61
Sin_R_8Term	40	64
Cos_R_4Term	4F	79
Cos_R_5Term	52	82
Cos_R_6Term	55	85
Coa_R_7Term	58	88
Cos_R_8Term	5B	91
Tan_R_8Term	2B	43
Arcsin_R_5Term	22	34
Arcsin_R_6Term	26	38
Arcsin_R_7Term	2A	42
Arcsin_R_8Term	2E	46
Arcoa_R_5Term	29	41
Arcos_R_6Term	2D	45
Arcos_R_7Term	31	49
Arcoa_R_8Term	35	53
Arctan_R_4Term	31	49
Arctan_R_5Term	35	53
Arctan_R_6Term	39	57
Arctan_R_7Term	3D	61
Arctan_R_8Term	41	65
Alt_Arctan_R_4Term	1E	30
Alt_Arctan_R_5Term	22	34
Alt_Arctan_R_6Term	26	38
Alt_Arctan_R_7Term	2A	42
Alt_Arctan_R_8Term	2E	46
Mod_Sin_R_4Term	6B	107
Mod_Sin_R_5Term	73	115
Mod_Sin_R_6Term	7B	123
Mod_Sin_R_7Term	83	131
Mod_Sin_R_8Term	8B	139
Mod_Cos_R_4Term	76	118
Mod_Coa_R_5Term	7E	126
Mod_Cos_R_6Term	86	134
Mod_Coa_R_7Term	8E	142
Mod_Cos_R_8Term	96	150
Mod_Tan_R_4Term	14	20
Mod_Tan_R_5Term	14	20
Mod_Tan_R_6Term	14	20
Mod_Tan_R_7Term	14	20
Mod_Tan_R_8Term	14	20
Taylor Series (cont.)		
Degree Operations		
Sin_D_5Term	8C	140
Sin_D_6Term	4C	76
Sin_D_7Term	50	80
Sin_D_8Term	54	84
Cos_D_5Term	9D	157
Cos_D_6Term	53	83
Cos_D_7Term	56	86
Cos_D_8Term	59	89
Tan_D_8Term	36	54
Mod_Sin_D_4Term	76	118
Mod_Sin_D_5Term	7E	126
Mod_Sin_D_6Term	86	134
Mod_Sin_D_7Term	8E	142
Mod_Sin_D_8Term	96	150
Mod_Cos_D_4Term	74	116
Mod_Cos_D_5Term	7C	124
Mod_Cos_D_6Term	84	132
Mod_Cos_D_7Term	8C	140
Mod_Cos_D_8Term	94	148
Mod_Tan_D_4Term	14	20
Mod_Tan_D_5Term	14	20
Mod_Tan_D_6Term	14	20
Mod_Tan_D_7Term	14	20
Mod_Tan_D_8Term	14	20

time of a function against the absolute precision of that function's results. Both the time and precision data are taken over the function argument domains listed at the bottom of each figure.

The domain specifications are of particular importance since a given function, apparently superior in terms of performance, may nevertheless operate correctly only over a small domain. This is, for example, true in the case of the radian arctangent benchmarks (Figures A-6 and A-12) where the "Alt Taylor" method appears to be the best performer. However, referring to the domain specification, it becomes apparent that the "Alt Taylor" method only provides the indicated performance over the domain [0.0, 0.4]. Other functions provide a more acceptable domain at a slightly higher throughput cost.

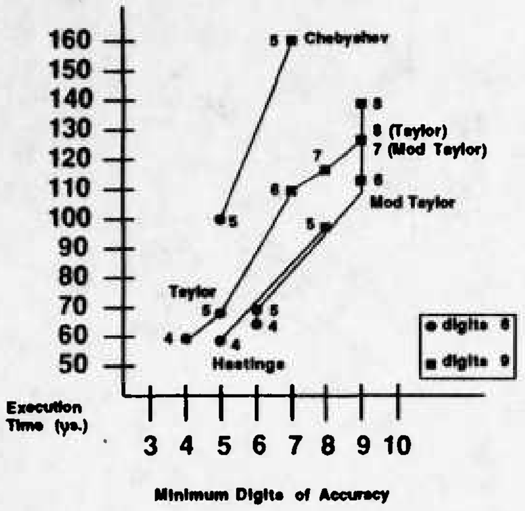
The absence of separate data for six and nine digit instantiations in the figures based on System B is due to the fact that compiler B always uses 1750A extended precision (approximately 9 decimal digits) to represent any generic floating-point object. Identical object code is used for each instantiation of a generic floating-point subprogram and, indeed, compiler B shares one object code instruction section among all instantiations of a given generic. The use of this "single copy" method implies that the running times of different instantiations of the same generic subprogram will be identical, regardless of the precision of the floating-point variables. Thus, the nine-digit worst case data applies for both six and nine-digit instantiations.

3. INTEGRATED EXECUTION BENCHMARK DATA

The integrated execution benchmarks, which integrate numerous CAMP parts, were run on the 32-bit minicomputer (System A above, Tables A-3, A-4, and A-5).

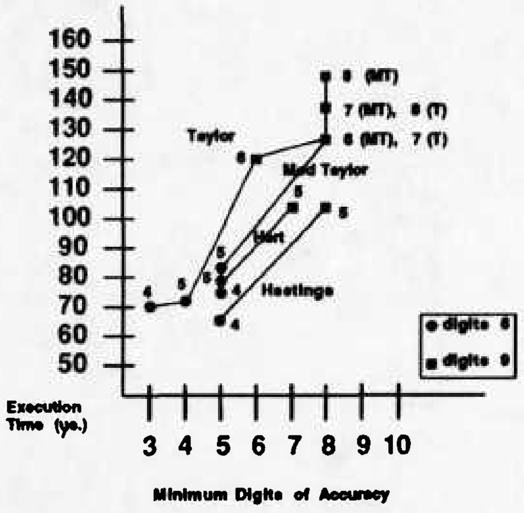
Standard output data for these benchmarks is supplied in the form of files accompanying the benchmark suite. This data can be used to verify that a compiler and target machine combination produce correct output for the benchmarks. The data is not reproduced here because it is quite lengthy and is not formatted for inclusion in a document.

Time-consumption data on the integrated execution benchmarks was automatically collected at run-time by the benchmark drivers. This data is presented in Tables A-3, A-4, and A-5.



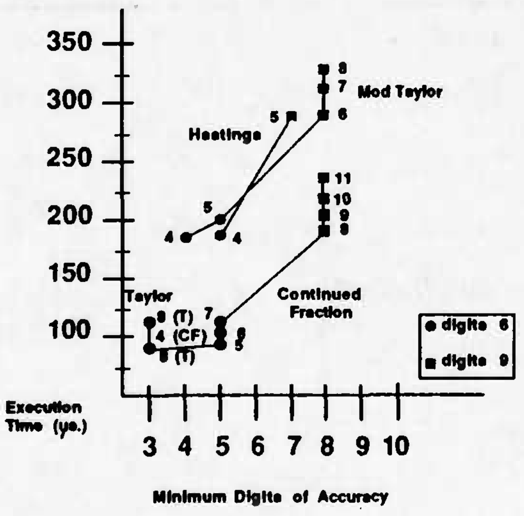
Numbers shown indicate the number of terms used in the polynomial expansion. The function domain over which this data applies is [-pi/2, pi/2].

Figure A-1. Radian Sine on System A



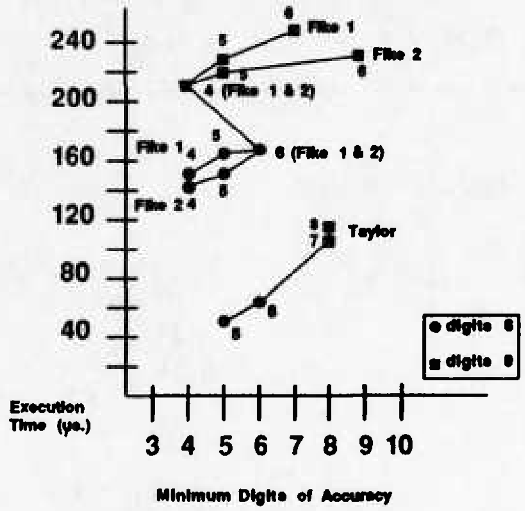
Numbers shown indicate the number of terms used in the polynomial expansion. The function domain over which this data applies is [0, pi].

Figure A-2. Radian Cosine on System A



Numbers shown indicate the number of terms used in the polynomial expansion. The function domain over which this data applies is [-1, 1].

Figure A-3. Radian Tangent on System A

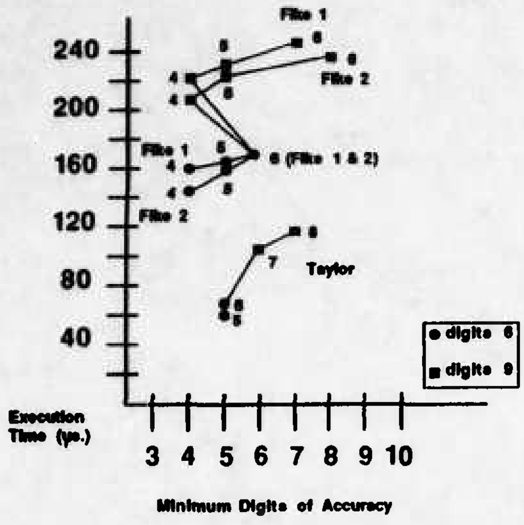


Numbers shown indicate the number of terms used in the polynomial expansion. The function domains over which these data apply are as follows:

Taylor Radian:	[-0.44, 0.44]
Fike 1 Semicircle:	[-0.8, 0.8]
Fike 2 Semicircle:	[-1.0, 1.0]

Where Fike 1 uses the Newton-Raphson square root and Fike 2 uses the Modified Newton-Raphson square root.

Figure A-4. Arcsine on System A

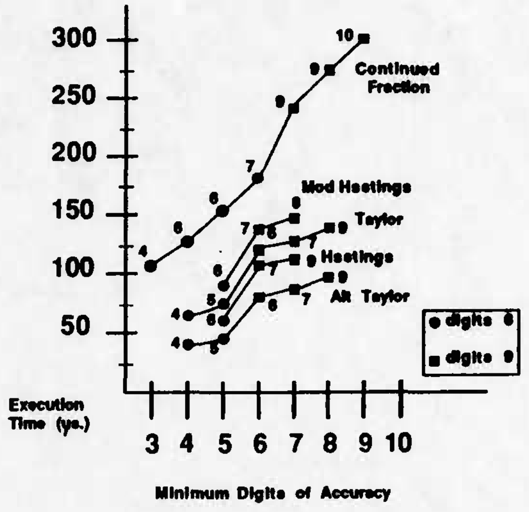


Numbers shown indicate the number of terms used in the polynomial expansion. The function domains over which these data apply are as follows:

Taylor Radian:	[-0.46, 0.46]
Fike 1 Semicircle:	[-0.6, 0.6]
Fike 2 Semicircle:	[-1.0, 1.0]

Where Fike 1 uses the Newton-Raphson square root and Fike 2 uses the Modified Newton-Raphson square root.

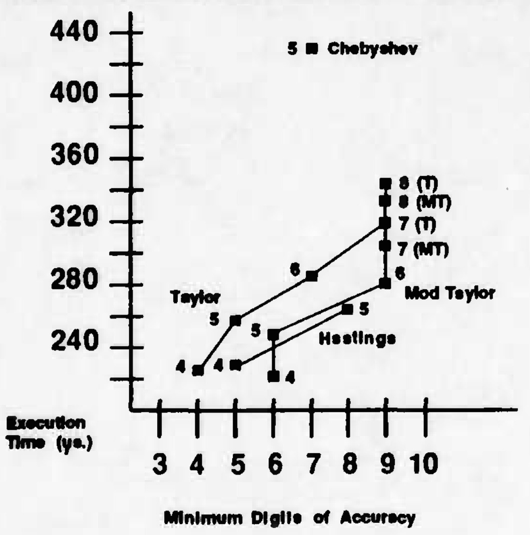
Figure A-5. Arccosine on System A



Numbers shown indicate the number of terms used in the polynomial expansion. The function domains over which these data apply are as follows:

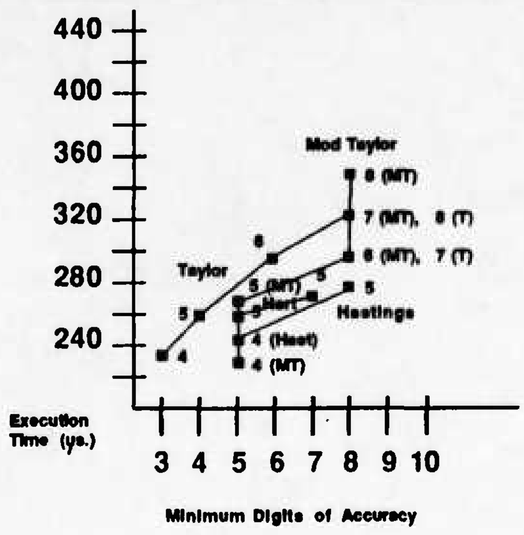
Taylor:	[2.63, 80.0]
Alt Taylor:	[0.0, 0.4]
Hastings:	[-1.0, 1.0]
Mod Hastings:	[0.0, 80.0]
Continued Fraction:	[0.0, 0.7167]

Figure A-6. Radian Arctangent on System A



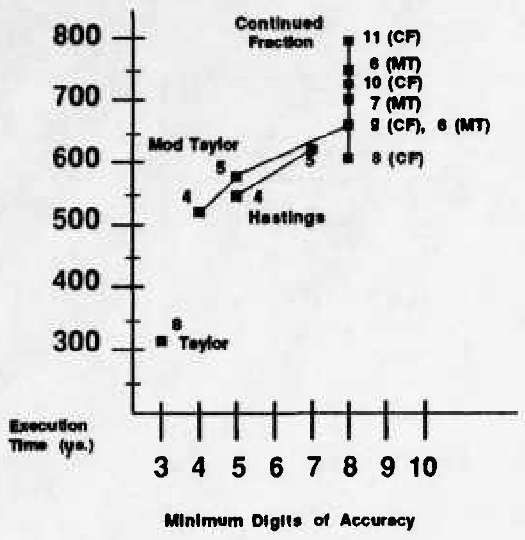
Numbers shown indicate the number of terms used in the polynomial expansion. The function domain over which this data applies is [-pi/2, pi/2].

Figure A-7. Radian Sine on System B



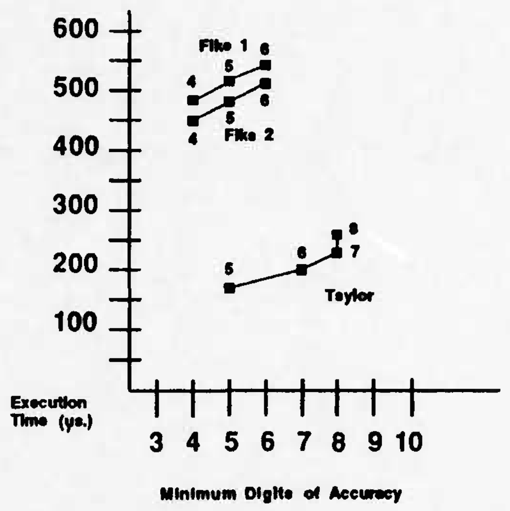
Numbers shown indicate the number of terms used in the polynomial expansion. The function domain over which this data applies is [0, pi].

Figure A-8. Radian Cosine on System B



Numbers shown indicate the number of terms used in the polynomial expansion. The function domain over which this data applies is [-1, 1].

Figure A-9. Radian Tangent on System B

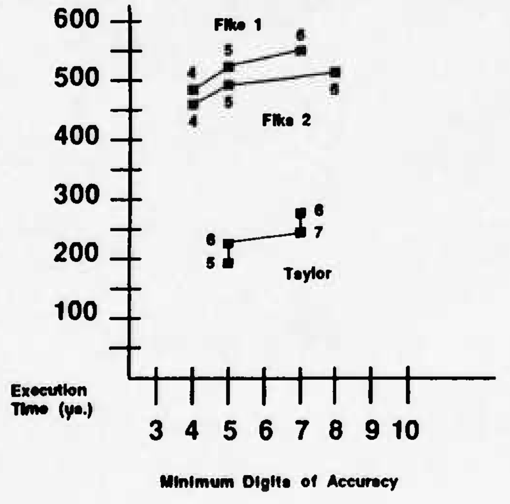


Numbers shown indicate the number of terms used in the polynomial expansion. The function domains over which these data apply are as follows:

Taylor Radian:	[-0.44, 0.44]
Fike 1 Semicircle:	[-0.6, 0.6]
Fike 2 Semicircle:	[-1.0, 1.0]

Where Fike 1 uses the Newton-Raphson square root and Fike 2 uses the Modified Newton-Raphson square root.

Figure A-10. Arcsine on System B

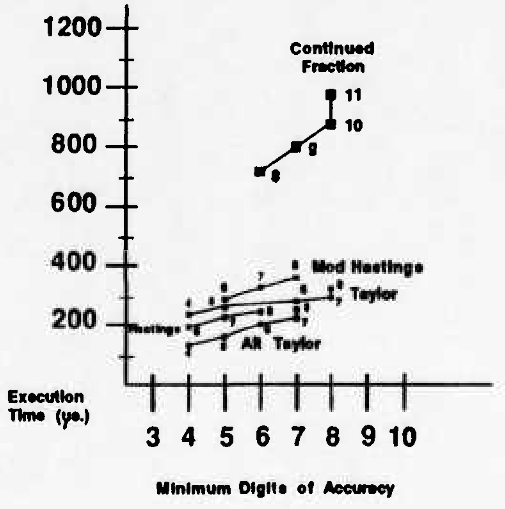


Numbers shown indicate the number of terms used in the polynomial expansion. The function domains over which these data apply are as follows:

Taylor Radian:	[-0.46, 0.46]
Fike 1 Semicircle:	[-0.6, 0.6]
Fike 2 Semicircle:	[-1.0, 1.0]

Where Fike 1 uses the Newton-Raphson square root and Fike 2 uses the Modified Newton-Raphson square root.

Figure A-11. Arccosine on System B



Numbers shown indicate the number of terms used in the polynomial expansion. The function domains over which these data apply are as follows:

Taylor:	[2.63, 50.0]
Alt Taylor:	[0.0, 0.4]
Hastings:	[-1.0, 1.0]
Mod Hastings:	[0.0, 50.0]
Continued Fraction:	[0.0, 0.7167]

Figure A-12. Radian Arctangent on System B

TABLE A-3. TIMING OF INTEGRATED EXECUTION 1

Integrated Execution 1 on VAX		
TLCSC Name (some names abbreviated) / LLCSC Name / Unit Name	Time (microsecs)	
	Per Call	Variation
Waypoint_Steering (P661)		
Compute_Turn_Angle_And_Direction	391.0	0
Compute_Turning_And_Nonturning_Dist	173.0	0
Distance_To_Current_Waypoint	409.0	0
Steering_Vector_Operations_W_Arcsin		
Initialize	5210.0	5
Update	2623.0	10
Turn_Test_Operations		
Stop_Test	62.0	2
Start_Test	61.0	0
Signal_Processing (P686)		
Absolute_Limiter		
Limit	43.0	0
Upper_Lower_Limiter		
Update_Limits	15.0	0
Limits	57.0	0

TABLE A-4. TIMING OF INTEGRATED EXECUTION 2

Integrated Execution 2 on VAX		
TLCSC Name (some names abbreviated) LLCSC Name Unit Name	Time (microsecs)	
	Per Call	Variation
Common_Navigation_Parts (P001)		
Update_Velocity		
Reinitialize	34.0	2
Current_Velocity	51.0	2
Update		
Compute_Ground_Velocity	509.0	2
Compute_Gravitational_Accel_Sin_Lat_In	332.0	2
Wander_Azimuth_Navigation_Parts (P002)		
Earth_Rotation_Rate		
Compute	338.0	2
Earth_Relative_Navigation_Rotat_Rates		
Compute	414.0	0
Total_Platform_Rotation_Rate	151.0	0
Compute_Latitude_Using_Two_Val_Arctan	623.0	0
Compute_Longitude_Using_Two_Val_Arctan	430.0	0
Compute_East_Velocity_With_Sin_Cos	231.0	2
Compute_North_Velocity_With_Sin_Cos	232.0	2
Compute_Coriolis_Acceleration	769.0	2
Compute_Wand_Azim_Angle_Two_Val_Arctan	432.0	0
Compute_Curvatures	820.0	0
Direction_Cosine_Matrix (P644)		
CNE_Operations		
Compute_First_Row_CNE_From_Ortho	199.0	0
CNE_Initialized_From_Reference	1647.0	2
Perform_Rect_Integration_Of_CNE	664.0	0
Reorthonormalize_CNE	1698.0	2
Aligned_CNE_Matrix	1178.0	2
General_Vector_Matrix_Algebra (P682)		
Matrix_Matrix_Multiply_Restricted	3861.0	0
General_Purpose_Math_Parts (P687)		
Accumulator		
Accumulate	19.0	2

TABLE A-5. TIMING OF INTEGRATED EXECUTION 3

Integrated Execution 3 on VAX		
TLCSC Name (some names abbreviated) / LLCSC Name / Unit Name	Time (microsecs)	
	Per Call	Variation
Kalman_Filter_Common_Parts (P651)		
State_Transition_And_Proc_Noise_Mat_Mgr		
Initialize	1603.0	1
Propagate	187290.0	10
Get_Current	113.0	1
Propagated_Phi	119.0	1
Error_Covariance_Matrix_Manager		
Initialize	69.0	1
Propagate	149540.0	10
P	119.0	1
State_Transition_Matrix_Manager		
Initialize	1547.0	1
Propagated_Phi	117.0	1
Propagate	28073.0	10
Kalman_Filter_Compact_H_Parts (P652)		
Compute_Kalman_Gains	10978.0	2
Update_Error_Covariance_Matrix	16267.0	2
Update_State_Vector	6360.0	2
Seq_Update_Cov_Matrix_And_State_Vector		
Update	67897.0	2
Kalman_Update		
Update	233355.0	2
Update_Error_Cov_Matrix_General_Form	73222.0	2
Kalman_Filter_Complicated_H_Parts (P653)		
Compute_Kalman_Gains	24687.0	1
Update_Error_Covariance_Matrix	60033.0	2
Update_State_Vector	12756.0	1
Seq_Update_Cov_Matrix_And_State_Vector		
Update	192150.0	10
Kalman_Update		
Update	361139.0	10
Update_Error_Cov_Matrix_General_Form	215098.0	10

APPENDIX B

ADA SOURCE CODE INVENTORY

The following tables comprise an inventory of all Ada source code used in the CAMP Armonics Benchmark Suite. In addition, the tables provide a cross-reference from the development name of a file to the ANSI name assigned to that file for transportation to other operating systems.

TABLE B-1. ADA SOURCE CODE INVENTORY
(1 of 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
CAMP Source Code	
001_000_COMMON_NAV_.ADA	A001000.ADA
001_001_COMMON_NAV.ADA	A001001.ADA
001_100_ALTITUDE_INTEGRATION.ADA	A001100.ADA
001_200_COMP_GROUND_VEL.ADA	A001200.ADA
001_300_COMP_GRAV_ACCEL_LAT_IN.ADA	A001300.ADA
001_400_COMP_GRAV_ACCEL_SIN_LAT_IN.ADA	A001400.ADA
001_500_COMP_HEADING.ADA	A001500.ADA
001_600_UPDATE_VELOCITY.ADA	A001600.ADA
001_700_SCALAR_VELOCITY.ADA	A001700.ADA
001_800_COMP_ROTATION_INCR.ADA	A001800.ADA
002_000_WA_NAV_.ADA	A002000.ADA
002_001_WA_NAV.ADA	A002001.ADA
002_100_EAST_VELOCITY.ADA	A002100.ADA
002_200_NORTH_VELOCITY.ADA	A002200.ADA
002_300_EARTH_REL_HOR_VELS.ADA	A002300.ADA
002_400_TOTAL_ANGULAR_VEL.ADA	A002400.ADA
002_500_CORIOLIS_ACCEL.ADA	A002500.ADA
002_600_CORIOLIS_ACCEL_TOT_RATES.ADA	A002600.ADA
002_700_RAD_OF_CURV.ADA	A002700.ADA
002_800_TOT_PLATFORM_ROT_RATE.ADA	A002800.ADA
002_900_EARTH_ROT_RATE.ADA	A002900.ADA
002_A00_EARTH_REL_ROT_RATE.ADA	A002A00.ADA
002_B00_LATITUDE.ADA	A002B00.ADA
002_C00_LATITUDE_USING_ATAN.ADA	A002C00.ADA
002_D00_LONGITUDE.ADA	A002D00.ADA
002_E00_WANDER_ANGLE.ADA	A002E00.ADA
002_F00_EAST_VEL_SIN_COS.ADA	A002F00.ADA
002_G00_NORTH_VEL_SIN_COS.ADA	A002G00.ADA
002_H00_EARTH_REL_HOR_VELS_SIN_COS.ADA	A002H00.ADA
002_I00_LATITUDE_USING_ATAN2.ADA	A002I00.ADA
002_J00_LONGITUDE_USING_ATAN2.ADA	A002J00.ADA
002_K00_WANDER_ANGLE_USING_ATAN2.ADA	A002K00.ADA
611_000_WGS72_METRIC_.ADA	A611000.ADA
613_000_WGS72_UNITLESS_.ADA	A613000.ADA
614_000_CONVERSION_FACTORS_.ADA	A614000.ADA
615_000_UNIVERSAL_CONSTANTS_.ADA	A615000.ADA
621_000_BDT_.ADA	A621000.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (2 OF 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
621_001_BDT.ADA	A621001.ADA
622_000_KDT_.ADA	A622000.ADA
622_001_KDT.ADA	A622001.ADA
634_000_CLOCK_HANDLER_.ADA	A634000.ADA
634_001_CLOCK_HANDLER.ADA	A634001.ADA
644_000_DCM_.ADA	A644000.ADA
644_001_DCM.ADA	A644001.ADA
651_000_KALMAN_COMMON_.ADA	A651000.ADA
651_001_KALMAN_COMMON.ADA	A651001.ADA
651_100_PHI_Q_MANAGER.ADA	A651100.ADA
651_200_P_MANAGER.ADA	A651200.ADA
651_300_PHI_MANAGER.ADA	A651300.ADA
652_000_KALMAN_COMPACT_.ADA	A652000.ADA
652_001_KALMAN_COMPACT.ADA	A652001.ADA
652_100_CKG.ADA	A652100.ADA
652_200_UPDATE_P.ADA	A652200.ADA
652_300_UPDATE_X.ADA	A652300.ADA
652_400_UPDATE_P_AND_X.ADA	A652400.ADA
652_500_KALMAN_UPDATE.ADA	A652500.ADA
652_600_UPDATE_P_GENERAL.ADA	A652600.ADA
653_000_KALMAN_COMPLICATED_.ADA	A653000.ADA
653_001_KALMAN_COMPLICATED.ADA	A653001.ADA
653_100_CKG.ADA	A653100.ADA
653_200_UPDATE_P.ADA	A653200.ADA
653_300_UPDATE_X.ADA	A653300.ADA
653_400_UPDATE_P_AND_X.ADA	A653400.ADA
653_500_KALMAN_UPDATE.ADA	A653500.ADA
653_600_UPDATE_P_GENERAL.ADA	A653600.ADA
661_000_WAYPOINT_STEERING_.ADA	A661000.ADA
661_001_WAYPOINT_STEERING.ADA	A661001.ADA
661_300_STEERING_VECTOR_OPNS.ADA	A661300.ADA
661_310_INITIALIZE.ADA	A661310.ADA
661_320_UPDATE.ADA	A661320.ADA
661_400_TURN_ANGLE_AND_DIRECTION.ADA	A661400.ADA
661_500_CRSSTRK_AND_HDG_ERR_OPNS.ADA	A661500.ADA
661_510_COMP_WHEN_TURNING.ADA	A661510.ADA
661_520_COMP_WHEN_NOT_TURNING.ADA	A661520.ADA
661_530_COMPUTE.ADA	A661530.ADA
661_600_DIST_TO_CURR_WAYPOINT.ADA	A661600.ADA
661_700_COMP_TURN_NONTURN_DIST.ADA	A661700.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (3 OF 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
661_800_TURN_TEST_OPNS.ADA	A661800.ADA
661_810_STOP_TEST.ADA	A661810.ADA
661_820_START_TEST.ADA	A661820.ADA
661_900_STEERING_VECTOR_OPNS_ARCSIN.ADA	A661900.ADA
661_A00_DIST_TO_CURR_WAYPOINT_ARCSIN.ADA	A661A00.ADA
681_000_C_ALGEBRA_.ADA	A681000.ADA
681_001_C_ALGEBRA.ADA	A681001.ADA
681_200_MATRIX_OPNS.ADA	A681200.ADA
681_230_SET_TO_IDENTITY_MATRIX.ADA	A681230.ADA
681_240_SET_TO_ZERO_MATRIX.ADA	A681240.ADA
681_400_MATRIX_SCALAR_OPNS.ADA	A681400.ADA
681_500_CROSS_PRODUCT.ADA	A681500.ADA
681_600_MATRIX_VECTOR_MULT.ADA	A681600.ADA
681_700_MATRIX_MATRIX_MULT.ADA	A681700.ADA
682_000_GENERAL_ALGEBRA_.ADA	A682000.ADA
682_001_GENERAL_ALGEBRA.ADA	A682001.ADA
682_100_VECTOR_OPNS_UC.ADA	A682100.ADA
682_200_MATRIX_OPNS_UC.ADA	A682200.ADA
682_300_DYN_SPARSE_MATRIX_UC.ADA	A682300.ADA
682_400_SYMM_HALF_STORAGE_MATRIX.ADA	A682400.ADA
682_500_SYMM_FULL_STORAGE_MATRIX_UC.ADA	A682500.ADA
682_600_DIAGONAL_MATRIX.ADA	A682600.ADA
682_700_VECTOR_SCALAR_OPNS_UC.ADA	A682700.ADA
682_800_MATRIX_SCALAR_OPNS_UC.ADA	A682800.ADA
682_900_DIAG_MATRIX_SCALAR_OPNS.ADA	A682900.ADA
682_A00_MATRIX_MATRIX_MULT_UR.ADA	A682A00.ADA
682_B00_MATRIX_VECTOR_MULT_UR.ADA	A682B00.ADA
682_C00_VECTOR_VECTOR_TRANS_MULT_UR.ADA	A682C00.ADA
682_D00_MATRIX_MATRIX_TRANS_MULT_UR.ADA	A682D00.ADA
682_E00_DOT_PRODUCT_OPN_UR.ADA	A682E00.ADA
682_F00_DIAG_FULL_MATRIX_ADD_UR.ADA	A682F00.ADA
682_G00_VECTOR_OPNS_C.ADA	A682G00.ADA
682_H00_MATRIX_OPNS_C.ADA	A682H00.ADA
682_J00_DYN_SPARSE_MATRIX_C.ADA	A682J00.ADA
682_K00_SYMM_FULL_STORAGE_MATRIX_C.ADA	A682K00.ADA
682_L00_VECTOR_SCALAR_OPNS_C.ADA	A682L00.ADA
682_M00_MATRIX_SCALAR_OPNS_C.ADA	A682M00.ADA
682_N00_MATRIX_MATRIX_MULT_R.ADA	A682N00.ADA
682_P00_MATRIX_VECTOR_MULT_R.ADA	A682P00.ADA
682_Q00_VECTOR_VECTOR_TRANS_MULT_R.ADA	A682Q00.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (4 OF 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
682_R00_MATRIX_MATRIX_TRANS_MULT_R.ADA	A682R00.ADA
682_S00_DOT_PRODUCT_OPN_R.ADA	A682S00.ADA
682_T00_DIAG_FULL_MATRIX_ADD_R.ADA	A682T00.ADA
682_U00_VECTOR_MATRIX_MULT_UR.ADA	A682U00.ADA
682_V00_VECTOR_MATRIX_MULT_R.ADA	A682V00.ADA
682_W00_ABA_TRANS_DSP_MATRIX_SQ_MATRIX.ADA	A682W00.ADA
682_X00_ABA_TRANS_VECTOR_SQ_MATRIX.ADA	A682X00.ADA
682_Y00_ABA_TRANS_VECTOR_SCALAR.ADA	A682Y00.ADA
682_Z00_COL_MATRIX_OPNS.ADA	A682Z00.ADA
683_000_STANDARD_TRIG_.ADA	A683000.ADA
683_001_STDTRIG_SYSFNS.ADA	A683001.ADA
684_000_GEOMETRIC_.ADA	A684000.ADA
684_001_GEOMETRIC.ADA	A684001.ADA
684_100_UNIT_RADIAL_VECTOR.ADA	A684100.ADA
684_200_UNIT_NL_VECTOR.ADA	A684200.ADA
684_300_SEG_UNIT_NL_VECTOR.ADA	A684300.ADA
684_400_GREAT_CIRCLE_ARC_LENGTH.ADA	A684400.ADA
684_500_SEG_UNIT_NL_VECTOR_ARCSIN.ADA	A684500.ADA
686_000_SIGNAL_.ADA	A686000.ADA
686_001_SIGNAL.ADA	A686001.ADA
686_100_UL_LIMITER.ADA	A686100.ADA
686_200_U_LIMITER.ADA	A686200.ADA
686_300_L_LIMITER.ADA	A686300.ADA
686_400_ABS_LIMITER.ADA	A686400.ADA
686_500_ABS_LIMITER_W_FLAG.ADA	A686500.ADA
686_600_FIRST_ORDER_FILTER.ADA	A686600.ADA
686_700_TUSTIN_LAG_FILTER.ADA	A686700.ADA
686_800_TUSTIN_LEAD_LAG_FILTER.ADA	A686800.ADA
686_900_SECOND_ORDER_FILTER.ADA	A686900.ADA
686_A00_TUSTIN_INTEGRATOR_W_LIMIT.ADA	A686A00.ADA
686_B00_TUSTIN_INT_W_ASYM_LIMIT.ADA	A686B00.ADA
687_000_GP_MATH_.ADA	A687000.ADA
687_001_GP_MATH.ADA	A687001.ADA
687_100_LOOKUP_EVEN.ADA	A687100.ADA
687_200_LOOKUP_UNEVEN.ADA	A687200.ADA
687_300_INCREMENTOR.ADA	A687300.ADA
687_400_DECREMENTOR.ADA	A687400.ADA
687_500_RUN_AVG.ADA	A687500.ADA
687_600_ACCUM.ADA	A687600.ADA
687_700_CHANGE_ACCUM.ADA	A687700.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (5 OF 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
687_800_CHANGE_CALC.ADA	A687800.ADA
687_900_INTEGRATOR.ADA	A687900.ADA
687_A00_INTERPOLATE.ADA	A687A00.ADA
687_D00_RSOS.ADA	A687D00.ADA
687_E00_SIGN.ADA	A687E00.ADA
687_F00_MEAN_VAL.ADA	A687F00.ADA
687_G00_MAD.ADA	A687G00.ADA
687_H00_LOOKUP_TWOWAY.ADA	A687H00.ADA
688_001_POLYNOMIALS.ADA	A688001.ADA
688_200_CHEBYSHEV.ADA	A688200.ADA
688_210_RADIAN_OPERATIONS.ADA	A688210.ADA
688_220_DEGREE_OPERATIONS.ADA	A688220.ADA
688_230_SEMICIRCLE_OPERATIONS.ADA	A688230.ADA
688_300_FIKE.ADA	A688300.ADA
688_310_SEMICIRCLE_OPERATIONS.ADA	A688310.ADA
688_400_HART.ADA	A688400.ADA
688_410_RADIAN_OPERATIONS.ADA	A688410.ADA
688_420_DEGREE_OPERATIONS.ADA	A688420.ADA
688_500_HASTINGS.ADA	A688500.ADA
688_510_RADIAN_OPERATIONS.ADA	A688510.ADA
688_520_DEGREE_OPERATIONS.ADA	A688520.ADA
688_800_MOD_NEWTON_RAPHSON.ADA	A688800.ADA
688_900_NEWTON_RAPHSON.ADA	A688900.ADA
688_A00_TAYLOR_SERIES.ADA	A688A00.ADA
688_A10_RADIAN_OPERATIONS.ADA	A688A10.ADA
688_A20_DEGREE_OPERATIONS.ADA	A688A20.ADA
688_A40_NATURAL_LOG.ADA	A688A40.ADA
688_A50_BASE_LOG.ADA	A688A50.ADA
688_B00_GENL_POLYNOMIAL.ADA	A688B00.ADA
688_C00_SYSTEM_FUNCTIONS.ADA	A688C00.ADA
688_C10_RADIAN_OPNS.ADA	A688C10.ADA
688_C20_SEMICIRCLE_OPNS.ADA	A688C20.ADA
688_C30_DEGREE_OPNS.ADA	A688C30.ADA
688_C40_SQUARE_ROOT.ADA	A688C40.ADA
688_C50_BASE_10.ADA	A688C50.ADA
688_C60_BASE_N.ADA	A688C60.ADA
688_D00_CONTINUED_FRACTIONS.ADA	A688D00.ADA
688_D10_RADIAN_OPERATIONS.ADA	A688D10.ADA
688_E00_CODY_WAITE.ADA	A688E00.ADA
688_E40_NATURAL_LOG.ADA	A688E40.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (6 OF 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
688_E50_BASE_N.ADA	A688E50.ADA
688_F00_REDUCTION.ADA	A688F00.ADA
851_000_UNIT_CONVERSION_.ADA	A851000.ADA
851_001_UNIT_CONVERSION.ADA	A851001.ADA
890_000_QUATERNION_.ADA	A890000.ADA
890_001_QUATERNION.ADA	A890001.ADA
890_100_EULER.ADA	A890100.ADA
890_200_NORMALIZED.ADA	A890200.ADA
11th Missile Code (some modified)	
BARO_ALT_FOR_KF_TESTS.ADA	MBAFKFT.ADA
BARO_TEST_DRIVER.ADA	MBROTES.ADA
DATA_RETRIEVAL_FOR_GUIDOPNS_TEST.ADA	MDTARET.ADA
DO_SUM_BARO_ALTIMETER_FOR_BIAS_TEST.ADA	MDSUMBA.ADA
DUMMY_AM.ADA	MDMMYAM.ADA
DUMMY_DO_SUM_BARO.ADA	MDMMYDO.ADA
DUMMY_INITIALIZE_NAVIGATOR.ADA	MDMMYIN.ADA
DUMMY_VELOCITY_COMPUTATIONS.ADA	MDMMYVE.ADA
EARTH_TO_BODY_TRANSFORM.ADA	MERTHTO.ADA
ENVIRONMENT_FOR_KF_TESTS.ADA	MEVIRON.ADA
EXECUTE_NAVIGATOR.ADA	MXNAVIG.ADA
EXECUTE_NAVIGATOR_TEST.ADA	MEXECUT.ADA
EX_NAV_KALMAN_FILTER_STUB.ADA	MEXNAVK.ADA
GUID_COMPUTER_FOR_GUIDOPNS_TEST.ADA	MGUIDCO.ADA
INCORPORATE_KALMAN_CORR.ADA	MINCORP.ADA
INTERNAL_BUS_BROADCAST_FOR_KF_TESTS.ADA	MINTERN.ADA
ISA_FOR_KF_TESTS.ADA	MISAFOR.ADA
KALMAN_FILTER_STUB.ADA	MKALMAN.ADA
M007_100_GUIDANCE_OPNS.ADA	M007100.ADA
M007_110_PROCESSOR_MODIFIED.ADA	M007110.ADA
M007_111_PRINCIPAL_VALUE.ADA	M007111.ADA
M007_112_PERFORM_INIT.ADA	M007112.ADA
M007_113_WAYPT_CNTRL_OPNS.ADA	M007113.ADA
M007_114_FLIGHT_CONTROL.ADA	M007114.ADA
M007_115_FIRST_ORDER.ADA	M007115.ADA
M012_000_GUIDANCE_DATA_TYPES_.ADA	M012000.ADA
M012_001_GUIDANCE_DATA_TYPES.ADA	M012001.ADA
M013_000_MISSION_DATA_.ADA	M013000.ADA
M014_000_NAV_COMPUTER_DATA_TYPES_.ADA	M014000.ADA
M014_001_NAV_COMPUTER_DATA_TYPES.ADA	M014001.ADA
M015_001_NAVIGATION_OPERATIONS.ADA	M015001.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (7 OF 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
M015_0200_EXECUTE_NAVIGATOR.ADA	M015020.ADA
M015_0900_SLAVE_CNE.ADA	M015090.ADA
M015_0C00_BARO_LOOP_COMPUTATIONS.ADA	M0150C0.ADA
M015_0H00_NAV_OPS_TEST_CODE.ADA	M0150H0.ADA
M017_000_ALIGNMENT_MEASUREMENTS_.ADA	M017000.ADA
M018_000_NAV_SYSTEM_.ADA	M018000.ADA
M019_000_KALMAN_TYPES_.ADA	M019000.ADA
M019_001_KALMAN_TYPES.ADA	M019001.ADA
M019_0100_F_OPERATIONS.ADA	M019010.ADA
M019_0200_PHI_OPERATIONS.ADA	M019020.ADA
M019_0800_ACTIVE_KHPO.ADA	M019080.ADA
M019_0900_PASSIVE_KHPO.ADA	M019090.ADA
M019_0A00_DOPPLER_KHPO.ADA	M0190A0.ADA
M021_000_KALMAN_FILTER_.ADA	M021000.ADA
M022_000_ENVIRONMENT_.ADA	M022000.ADA
M024_000_H_ROW_.ADA	M024000.ADA
M024_001_H_ROW.ADA	M024001.ADA
M611_000_WGS72_METRIC_.ADA	M611000.ADA
M612_000_WGS72_ENGINEERING_.ADA	M612000.ADA
MEASUREMENTS_FOR_KF_TESTS.ADA	MMEASUR.ADA
MESSAGE_MANAGER_FOR_GUIDOPNS_TEST.ADA	MMESSAG.ADA
MISSION_DATA_FOR_GUIDOPNS_TEST.ADA	MMISSIO.ADA
NAVIGATION_OPERATIONS_.ADA	MNAVIGA.ADA
NAV_SYSTEM_STUB.ADA	MNAVSYS.ADA
OCU_FOR_KF_TESTS.ADA	MOCUFOR.ADA
SCP_FOR_KF_TESTS.ADA	MSCPFOR.ADA
TLM_FOR_KF_TESTS.ADA	MTLMFOR.ADA
VELOCITY_COMPUTATIONS.ADA	MVELOCI.ADA
VELOCITY_COMPUTATIONS_TEST.ADA	MVELOCT.ADA
VEL_TEST_DRIVER.ADA	MVELTES.ADA
WANDER_ANGLE_COMPUTATIONS.ADA	MWANDER.ADA
Compilation Benchmark Source Code	
10_WGS72U_.ADA	C10WGS7.ADA
20_NPNAV_.ADA	C20NPNA.ADA
21_NPNAV.ADA	C21NPNA.ADA
30_KFCOMMON_.ADA	C30KFCO.ADA
31_KFCOMMON.ADA	C31KFCO.ADA
40_KFCOMPLICATED_.ADA	C40KFCO.ADA
41_KFCOMPLICATED.ADA	C41KFCO.ADA
50_POLY_.ADA	C50POLY.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (8 OF 10)

Amonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
51_POLY.ADA	C51POLY.ADA
60_GVMA_.ADA	C60GVMA.ADA
61_GVMA.ADA	C61GVMA.ADA
70_GPMATH_.ADA	C70GPMA.ADA
71_GPMATH.ADA	C71GPMA.ADA
80_CVMA_.ADA	C80CVMA.ADA
81_CVMA.ADA	C81CVMA.ADA
90_STDTRIG_.ADA	C90STDT.ADA
91_STDTRIG.ADA	C91STDT.ADA
A0_GEO_.ADA	CA0GEOX.ADA
A1_GEO.ADA	CA1GEOX.ADA
B0_UNIVCONST_.ADA	CB0UNIV.ADA
C0_CONVFACTORS_.ADA	CC0CONV.ADA
D0_BDT_.ADA	CD0BDTX.ADA
D1_BDT.ADA	CD1BDTX.ADA
E0_WPS_.ADA	CE0WPSX.ADA
E1_WPS.ADA	CE1WPSX.ADA
F0_WGS72_.ADA	CF0WGS7.ADA
G0_KDT_.ADA	CG0KDTX.ADA
G1_KDT.ADA	CG1KDTX.ADA
Z1_NP_TDRVR.ADA	CZ1NPTD.ADA
Z2_WPS_TDRVR.ADA	CZ2WPST.ADA
Z3_KF_TDRVR.ADA	CZ3KFTD.ADA
Original Benchmark Source Code	
683A_000_STANDARD_TRIG_.ADA	A683A00.ADA
683A_001_STDTRIG_FIKE_HASTINGS.ADA	A683A00.ADA
683B_000_STANDARD_TRIG_.ADA	A683B00.ADA
683B_001_STDTRIG_FIKE_HASTINGS.ADA	A683B00.ADA
683_002_STD_TRG_NOSYS.ADA	A683002.ADA
687_C01_NEWTON_SQRT.ADA	A687C01.ADA
688_000_POLYNOMIALS_.ADA	A688000.ADA
688_310_SEMICIRCLE_OPERATIONS.ADA	A688310.ADA
ANALYZE.ADA	BANALYZ.ADA
BENCHMARKING_TOOLS.ADA	BBNMARK.ADA
BENCHMARKING_TOOLS_.ADA	BBNCHMA.ADA
BENCHMARK_CONTENTS.ADA	BBNCHMR.ADA
BENCHMARK_CONTENTS_.ADA	BBNCHMK.ADA
CHEBYSHEV6_DRIVER.ADA	BCHEBY6.ADA
CHEBYSHEV9_DRIVER.ADA	BCHEBY9.ADA
CODY6_DRIVER.ADA	BCDY6DR.ADA

TABLE B-1. ADA SOURCE CODE INVENTORY (9 OF 10)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	ANSI Name
CODY9_DRIVER.ADA	BCDY9DR.ADA
CONTINUED6_DRIVER.ADA	BCNT6DR.ADA
CONTINUED9_DRIVER.ADA	BCNT9DR.ADA
CONTINUED_FRACTION_BENCHMARK.ADA	BCNTFRA.ADA
CONTINUED_FRACTION_BENCHMARK_.ADA	BCNTFRC.ADA
CPU_CLOCK.ADA	BCPUCLO.ADA
FIKE6_DRIVER.ADA	BFIKE6D.ADA
FIKE9_DRIVER.ADA	BFIKE9D.ADA
HART6_DRIVER.ADA	BHART6D.ADA
HART9_DRIVER.ADA	BHART9D.ADA
HASTINGS6_DRIVER.ADA	BHAST6D.ADA
HASTINGS9_DRIVER.ADA	BHAST9D.ADA
INT_BENCHMARKING_TOOLS.ADA	BINTBEN.ADA
INT_BENCHMARKING_TOOLS_.ADA	BINTBNC.ADA
KALMAN_COMMON_TEST.ADA	BKALMNC.ADA
KALMAN_COMMON_TEST_.ADA	BKALMAN.ADA
KALMAN_COMPACT_DRIVER.ADA	BKLMANC.ADA
KALMAN_COMPACT_TEST.ADA	BKLMNCO.ADA
KALMAN_COMPACT_TEST_.ADA	BKLMCOM.ADA
KALMAN_COMPLICATED_DRIVER.ADA	BKLMNCM.ADA
KALMAN_COMPLICATED_TEST.ADA	BKLNCOM.ADA
KALMAN_COMPLICATED_TEST_.ADA	BKLMCOM.ADA
MATRIX_OUTPUT.ADA	BMATRIX.ADA
MATRIX_OUTPUT_.ADA	BMTRIXO.ADA
NEWTON6_DRIVER.ADA	BNWTN6D.ADA
NEWTON9_DRIVER.ADA	BNEWTN9.ADA
POLYNOMIALS_NO_SYS_FUNC.ADA	BPLYNOM.ADA
POLYNOMIALS_NO_SYS_FUNC_.ADA	BPOLYNO.ADA
POLYNOMIAL_BENCHMARK.ADA	BPOLYNM.ADA
POLYNOMIAL_BENCHMARK_.ADA	BPOLNOM.ADA
REDUCE_SIM_LOG.ADA	BREDUCE.ADA
SYSTEM_DRIVER.ADA	BSYSTEM.ADA
TAYLOR6_DEGREE_DRIVER.ADA	BTYLOR6.ADA
TAYLOR6_RADIAN_DRIVER.ADA	BTAYLR6.ADA
TAYLOR9_DEGREE_DRIVER.ADA	BTYLOR9.ADA
TAYLOR9_RADIAN_DRIVER.ADA	BTAYLR9.ADA
Benchmark VAX/VMS Command Procedures	
ACT_COMPILATION_RUN.COM	JACTCOM.COM
COMPILATION_BENCHMARKS.COM	JCMPILA.COM
COMPILE_BENCHMARK_SUPPORT.COM	JCOMPIL.COM

TABLE B-1. ADA SOURCE CODE INVENTORY (CONCLUDED)

Armonics Benchmark Inventory and Cross-Reference	
Development Name	**ANSI Name**
COMPILE_TOOLS.COM	JCMPLTO.COM
INT_EXEC1_COM_LINK.COM	JINT1CM.COM
INT_EXEC2_COM_LINK.COM	JINT2CM.COM
INT_EXEC3_COM_LINK.COM	JINT3CM.COM
MODIFIED_POLY6_COM_LINK.COM	JMDPOL6.COM
MODIFIED_POLY9_COM_LINK.COM	JMDPOL9.COM
POLY6_COM_LINK.COM	JPLY6CM.COM
POLY9_COM_LINK.COM	JPLY9CM.COM
SYSTEM_COM_LINK.COM	JSYSCML.COM
TLD_BENCHMARKS_COM_LINK.COM	JTLDBCO.COM
TLD_COMPILATION_RUN.COM	JTLDCOM.COM
VAX_ANALYZE_COM_LINK.COM	JVAXANL.COM
VAX_ANALYZE_POLY.COM	JVAXALY.COM
VAX_BENCHMARKS_COM_LINK.COM	JVAXBSC.COM
VAX_COMPILATION_RUN.COM	JVAXCOM.COM
VAX_INT_EXEC1_RUN.COM	JVAXI1R.COM
VAX_INT_EXEC2_RUN.COM	JVAXI2R.COM
VAX_INT_EXEC3_RUN.COM	JVAXI3R.COM
VAX_POLY_RUN.COM	JVAXPRU.COM
ANSI/Development Name Conversion	
ANSI2DV.COM	ANSI2DV.COM
DV2ANSI.COM	DV2ANSI.COM
Standard Output Data Files	
VAX_INT_EXEC1_RUN.DAT	DVXIE1R.DAT
VAX_INT_EXEC2_RUN.DAT	DVXIE2R.DAT
VAX_INT_EXEC3_RUN.DAT	DVXIE3R.DAT
HART6_DRIVER.ANA	DHART6D.ANA
HART6_DRIVER.DAT	DHART6D.DAT

INITIAL DISTRIBUTION LIST

GTE GOVERNMENT SYS CORP	1
ADVANCED DIGITAL SYSTEMS	1
AFATL/FXG	4
MILITARY COMPUTER SYSTEMS	1
LOCKHEED/0/62-81, B/563, F15	1
HUGHES/FULLERTON	1
UNISYS/MS-E1D08	1
WESTINGHOUSE/BALTIMORE	1
AFWAL/AAAS-2	1
BOOZ-ALLEN & HAMILTON, INC	1
BOEING AEROSPACE COMPANY/MS 8H-09	1
BOEING AEROSPACE CO	1
AD/YGE	1
SOFTWARE PRODUCTIVITY CONSORTIUM	5
ARMY CECOM/AMSEL-COM-IA	1
NAVAL TRAINING SYS CENTER/CODE 251	1
SCIENCE APPLICATIONS INTL CORP	1
RAYTHEON/MSL SYS DIVISION	1
CALSPAN	1
KAMAN SCIENCES CORPORATION	1
NAVAL RESEARCH LAB/CODE 5595	1
CARNEGIE MELLON UNIV/SEI/SHOLOM	1
COLEMAN RESEARCH CORP	1
COLSA, INC	1
CONTROL DATA CORPORATION	1
WINTEC	1
CONTROL DATA/DEPT 1855	1
DACS/RADC/COED	1
RAYTHEON/EQPT DIV	1
BMO/ACD	1
DDC-I, INC	1
ENGINEERING & ECONOMICS RESEARCH/ DIV OFFICE	1
BDM CORP	1
AFATL/FXG/EVERS	1
ESD/SYW-JPMO	1
FORD AEROSPACE & COMM CORP/MS H04	1
UNIV OF COLORADO #202	1
ANALYTICS	1
AFWAL/FIGL	1
WESTINGHOUSE ELECTRIC CORP/MS 5220	1
GENERAL DYNAMICS/MZ W2-5530	1
HONEYWELL INC	1
TAMSCO	1
STARS	1
FORD AEROSPACE/MS 2/206	1
GRUMMAN HOUSTON CORPORATION	1
NAVAL AVIONICS CENTER/NAC-825	1
NASA JOHNSON SPACE CENTER/EH/GHG	1
BOEING AEROSPACE/MS-8Y97	1
HARRIS CORPORATION/GISD	1
CARNEGIE MELLON UNIV/ SOFTWARE ENGINEERING INST	1
NOAA/ERL/R/E/AL4	1
INTERMETRICS, INC/G. RENTH	1
INTERMETRICS, INC/D.P. SMITH	1
FORD AEROSPACE/WEST DEVEL DIV	1
AD/ENE	1
ROCKWELL/MS-GA21	1
GRUMMAN CORP/MS D-31-237	1
INSTITUTE OF DEFENSE ANALYSIS	1
TELEDYNE BROWN/MS 178	1
USAF/TAWC/SCAM	1
BOEING AEROSPACE CO/D. LINDBERG	1
LOGICON	1
EASTMAN KODAK/DEPT 47	1
SYSTEMS CONTROL TECH, INC	1
E-SYSTEMS/GARLAND DIV	1
AFWAL/AAAF	1
MARTIN DEVELOPMENT	1
MA COMPUTER ASSOCIATES INC	1
IBM FEDERAL SYS DIV/MC 3206C	1
MCDONNELL DOUGLAS/INCO, INC	1
UNITED TECH, ADVANCED SYS	1
MCDONNELL AIRCRAFT CO/DEPT 300	1
WESTINGHOUSE ELEC/MS 432	1
MHP FU-TECH, INC	1
ITT AVIONICS	1
COSMIC/UNIV OF GA	1
NAVAL OCEAN SYS CENTER/CODE 423	1
NAVAL WEAPONS CTR/CODE 3922	1
ODYSSEY RESEARCH ASSOCIATES, INC	1
USA ELEC PROVING GRD/STEEP MT-DA	1
PATHFINDER SYS	1
BDM CORPORATION	1
PERCEPTRONICS, INC	1
PHOENIX INTERNATIONAL	1
MCDONNELL DOUGLAS ASTRO CO	1
GTE LABORATORY/RUBEN PRIETO-DIAZ	1
PROPRIETARY SOFTWARE SYSTEMS	1
ADVANCED TECHNOLOGY	8
STANFORD TELECOMMUNICATIONS, INC	1
RATIONAL	1
LOCKHEED MISSILES & SPACE CO	1
HERCULES DEFENSE ELEC SYS	1
AEROSPACE CORP	1
ROGERS ENGINEERING & ASSOCIATES	1
ADASOFT INC	1
ESD/XRSE	1
SANDERS/MER 24-1212	1
CSC/ERIC SCHACHT	1
COMPUTER TECH ASSOCIATES, INC	1

INITIAL DISTRIBUTION LIST (CONCLUDED)

SCIENCE APPLICATIONS INTER CORP	1
HQ CASE/CBRC	1
GOULD INC/CSD	1
HQ AFSPACECOM/LKWD/STOP 32	1
SVERDRUP/EGLIN	1
HONEYWELL INC/CLEARWATER	1
TECHNOLOGY SERVICE CORP	1
AEROSPACE/LOS ANGELES	1
SOFTWARE ARCHITECTURE & ENGIN	1
LORAL SYSTEMS GROUP/D/476-C2E	1
NADC/CODE 7033	1
UNISYS/PAOLA RESEARCH CTR	1
SIRIUS INC	1
GENERAL RESEARCH CORP	1
SOFTECH, INC/R.L. ZALKAN	1
SOFTECH, INC/R.B. QUANRUD	1
SOFTWARE CERTIFICATION INS	1
SOFTWARE CONSULTING SPECIALIST	1
SOFTWARE PRODUCTIVITY SOLUTIONS, INC	1
STAR-GLO INDUSTRIES INC	1
NADC/CODE 50C	1
WESTINGHOUSE/BALTIMORE	1
MITRE CORPORATION	1
SYSCON CORP/I. WEBER	1
SYSCON CORP/C. MORSE	1
SYSCON CORP/T. GROBICKI	1
AEROSPACE CORPORATION/M-8-026	1
TEXTRON DEFENSE SYSTEMS	1
GENERAL DYNAMICS/MZ 1774	1
TIBURON SYSTEMS, INC	1
TRW DEFENSE SYS GROUP	1
NASA SPACE STATION	1
BALLISTIC MSL DEF ADVANCED/ TECHNOLOGY CENTER	1
IBM CORPORATION/FSD	1
VISTA CONTROLS CORPORATION	1
VITRO CORPORATION	1
NAVAL RESEARCH LABORATORY/CODE 5150	1
CACI, INC	1
AFSC/PLR	5
DIRECTOR ADA JOINT PROGRAM OFFICE	1
MCDONNELL DOUGLAS ASTRONAUTICS/ E 434/106/2/MS22	7
SDIO/S/PI	1
ADVANCED SOFTWARE TECH SPECIALTIES	1
DTIC-DDAC	2
AFCSA/SAMI	1
AUL/LSE	1
FTD/SDNF	1
AFWAL/FIES/SURVIAC	1
HQ USAFE/INATW	1
AFATL/CC	1
AFATL/CA	1
AFATL/DOIL	2
6575 SCHOOL SQUADRON	1
IITRI	1

SUPPLEMENTARY

INFORMATION

DEPARTMENT OF THE AIR FORCE
WRIGHT LABORATORY (AFSC)
EGLIN AIR FORCE BASE, FLORIDA, 32542-5434

ERRATA

AD-B129570

REPLY TO ATTN OF: MNOI

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center
ATTN: DTIC/HAR (Mr William Bush)
Bldg 5, Cameron Station
Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

Technical Report Number	AD Number
1. 88-18-Vol-4	ADB 120 251
2. 88-18-Vol-5	ADB 120 252
3. 88-18-Vol-6	ADB 120 253
4. 88-25-Vol-1	ADB 120 309
5. 88-25-Vol-2	ADB 120 310
6. 88-62-Vol-1	ADB 129 568
7. 88-62-Vol-2	ADB 129 569
8. 88-62-Vol-3	ADB 129-570
9. 85-93-Vol-1	ADB 102-654
10. 85-93-Vol-2	ADB 102-655
11. 85-93-Vol-3	ADB 102-656
12. 88-18-Vol-1	ADB 120 248
13. 88-18-Vol-2	ADB 120 249
14. 88-18-Vol-7	ADB 120 254
15. 88-18-Vol-8	ADB 120 255
16. 88-18-Vol-9	ADB 120 256
17. 88-18-Vol-10	ADB 120 257
18. 88-18-Vol-11	ADB 120 258
19. 88-18-Vol-12	ADB 120 259

2. If you have any questions regarding this request call me at DSN 872-4620.

[illegible]
LYNN S. WARGO
Chief, Scientific and Technical
Information Branch

1 Atch
AFDTC/PA Ltr, dtd 30 Jan 92

ERRATA

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC)
EGLIN AIR FORCE BASE, FLORIDA 32542-5000

REPLY TO
ATTN OF: PA (Jim Swinson, 882-3931)

30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

Virginia N. Pribyla
VIRGINIA N. PRIBYLA, Lt Col, USAF
Chief of Public Affairs

AFDTC/PA 92-039