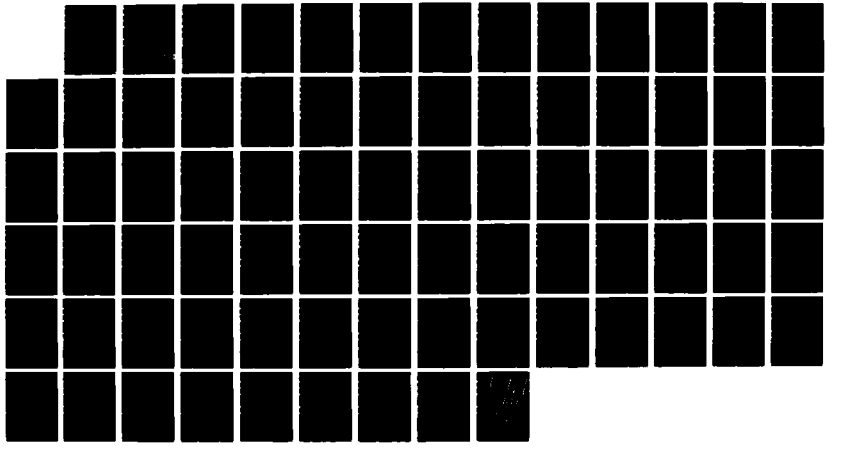
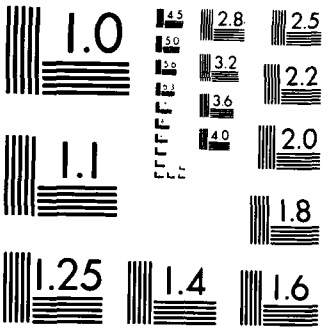
AD-A189 679 EXAMINATION OF THE EFFECTS OF USING ADA (TRADE NAME) IN 1/1
FLIGHTCCONTROL SOFTWARE(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI J D KLEMENS
UNCLASSIFIED DEC 87 AFIT/GCS/MA/87D-3 F/G 12/5 NL





EXAMINATION OF THE EFFECTS OF USING ADA® IN FLIGHT CONTROL SOFTWARE

THESIS

John D. Klemens, B.S.

Captain, USA

AFIT/GCS/MA/87D-3

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/MA/87D-3

EXAMINATION OF THE EFFECTS OF USING ADA® IN FLIGHT CONTROL SOFTWARE

THESIS

John D. Klemens, B.S.

Captain, USA

AFIT/GCS/MA/87D-3

Approved for public release; distribution unlimited

® Ada is a registered trademark of the U.S. Government (Ada Joint Program Office)

AFIT/GCS/MA/87D-3

EXAMINATION OF THE EFFECTS OF USING ADA® IN FLIGHT CONTROL SOFTWARE

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Information Systems)

John D. Klemens, B.S.

Captain, USA

December 1987

® Ada is a registered trademark of the U.S. Government (Ada Joint Program Office)

Preface

The research described in this thesis provides a set of benchmarks which would provide flight control software personnel with the ability to evaluate various Ada compilers, the compiler's run-time systems, and various MIL-STD-1750A processors for suitability for flight control software. The benchmarks developed during this thesis are designed to help the flight control and compiler performance evaluation communities.

I would like to thank Lieutenants Marc Pitarys and Bob Marmelstein for their help in getting access to the Ada compilers and 1750A processors as well as furnishing instructions and manuals on their use. I would also like to thank Captain Dan Joyce and Major Roger Kontak, fellow students, for their encouragement and help while we spent many hours in the Avionics Laboratory.

Table of Contents

List of Figures

List of Tables

List of Acronyms

ABICS-II	Ada Based Integrated Control System II
ACEC	Ada Compiler Evaluation Capability
ACM	Association for Computing Machinery
ACVC	Ada Compiler Validation Capability
AFTI	Advanced Fighter Technology Integration
AFWAL	Air Force Wright Aeronautical Labs
ANSI	American National Standards Institute
ATF	Advanced Tactical Fighter
DEC	Digital Equipment Coroporation
DFCS	Digital Flight Control System
DOD	Department of Defense
FLCC	Flight Control Computer
LEF	Leading Edge Flap
LRM	Language Reference Manual
OFP	Operational Flight Program
PIWG	Performance Issues Working Group
RTS	Run Time System
UCCS	University of Colorado at Colorado Springs
USAF	United States Air Force

Abstract

This research characterizes flight control software in terms of the Ada (TM) constructs used and of the performance characteristics that determine the suitability of a particular compiler/processor system for flight control software. A new set of three flight control software benchmarks based on this characterization was evaluated on two MIL-STD-1750A processors using two Ada compilers. Results show that compiler/processor combinations can be compared for their ability to implement flight control software in execution speed and memory size required. The benchmarks provide time and space requirements for a typical sample of flight control software.

I. Introduction

Statement of the Problem

The Department of Defense developed the Ada programming language in the 1970's to facilitate software development in defense applications, primarily embedded systems applications (Booch, 1983:3-4). Flight control is a specialized embedded system application which controls an aircraft as it travels through the atmosphere. Future United States Air Force (USAF) flight control software will be implemented in Ada rather than JOVIAL and assembly languages, the current languages. However, no determination has yet been made on the ability of the Ada language to implement flight control software efficiently and effectively.

Background

The JOVIAL programming language was developed in conjunction with the development of avionics and flight control applications. This joint evolution has made JOVIAL particularly well suited for flight control applications. Flight control software has often been a combination of JOVIAL and assembly languages (Joslin, 1987).

Both the machine dependency of assembly languages and the intermixing of JOVIAL and assembly languages, however, make flight control software hard to maintain and limit the portability of such software to other flight control systems. Because the Department of Defense (DOD) developed

Ada to improve the maintainability, reliability, and portability of software (Booch, 1983:3-4), DOD policy now requires that software developed for major systems be written in Ada (DOD, 1987).

The current DOD policy was preceded by numerous DOD Directives and Instructions, all of which augured the latest policy which requires the use of Ada (Witt, 1985:3-4). In accordance with these policies, the "Ada Based Integrated Control System II (ABICS-II)," the first attempt to use Ada for a major USAF weapon system, resulted in a majority of the flight control software for the F-15 being written in Ada (portions of the software were written in assembly language to increase the speed of execution) (Landy and others, 1986). This software, containing a mixture of Ada and assembly language, was tested in flight, identifying a number of constructs of Ada (see Table I) as important in flight control applications (Landy and others, 1986:9-2 to 9-15). The ABICS-II project was also one of the first applications integrating the various on-board computer controlled systems (e.g., flight and fire control systems); the trend today is toward increasing integration of computer controlled systems on aircraft (Joslin, 1987).

The ability of Ada to handle future real-time embedded applications effectively depends upon the efficiency of the generated code, the run-time system (RTS) furnished with the compiler, and the processor architecture. In flight

TABLE I. Ada Constructs Important in Flight Control (Landy and others, 1986:9-2 to 9-15)

Fixed and Floating Point Data Types
Subprograms
Conditional Control
Iterative Control
Packages
Generics
Tasking
Exceptions
Implementation-dependent Features (specifically representation specifications)
Pragmas
Low-level Input/Output

control applications, the speed of execution of the software can not only affect the performance of an aircraft, but even be a matter of life and death, so the ability to determine the compiler/1750A processor which generates the fastest Ada code can lead to safer and better performing aircraft. Also, since contemporary flight control computers are based on MIL-STD-1750A processors limited to 64K or 128K bytes of memory, a flight control compiler must also be space efficient. The memory limit is due to the addressing limitations of MIL-STD-1750A (DOD, 1980).

Current compiler assessment capabilities are divided into validation criteria and evaluation measurements. *Validation criteria* enumerate the features of a language which a compiler *must* implement to be used for software development for the validating agency. The validation criteria for the DOD are contained in a series of tests called the Ada Compiler

Validation Capability (ACVC). This test suite determines if a target compiler conforms with ANSI/MIL-STD-1815A, Ada Programming Language Reference Manual (LRM) (DOD, 1983), in which case it is considered *validated*. The ACVC does not provide data with which to compare compilers or test implementation dependent features.

Evaluation measurements determine the *efficiency* with which a compiler implements a language feature or features. The efficiency of a compiler can be measured in the speed (either compilation or execution) or the size of the machine code it generates. Two of the test suites used to evaluate Ada compilers are the prototype Ada Compiler Evaluation Capability (ACEC) (Hook and others, 1985) and the Performance Issues Working Group (PIWG) of the Association for Computing Machinery (ACM) benchmarks (Squire, 1986). The ability of the ACEC, PIWG, and other test suites to evaluate compiler performance for flight control applications has not been studied.

Scope

This research developed tests to evaluate the code size and execution speed of Ada code used to implement flight control software. The goal of the research was to obtain a comparative evaluation of Ada compilers for flight control software. The speed of execution of the Ada code was chosen as one evaluation criterion because the real-time embedded nature of flight control software requires operations to be

executed within a specified time frame for a particular flight control application. Execution speeds within this time frame would insure suitability of a tested compiler and processor for that application. The code size generated was evaluated as a second criterion because the limited memory available for flight control software using 1750A processors requires the code to be as space efficient as possible. A smaller code size will allow an application to include more finely detailed control law computations and better redundancy management schemes, resulting in more effective flight control software.

Limitations

This research did not attempt to obtain an absolute evaluation of compilers. An absolute evaluation would need to test all the language features that a compiler implements. Since flight control software does not use all features of Ada, an absolute comparison is not necessary. For example, text input/output is not necessary for flight control software.

Research Approach

The first step of this research was a definitional phase, to characterize flight control software in terms of Ada constructs used, algorithms employed, and flight control functions addressed. Next, Ada programs which embodied each

feature of the definition were coded, tested, and evaluated in an incremental fashion, with one program being evaluated before coding on the next was begun. This incremental approach allowed the research to be refined as it proceeded: after each evaluation, correctional feedback from the sponsor could be integrated and all aspects of the research could be judged for their suitability to the final goal. Suitability of a test was analyzed by surveying experts in the fields of flight control software, flight control mechanization, and compiler benchmarking on whether the test as designed would provide the results needed to answer the thesis problem. This approach limited the number of tests developed; however, it reduced useless activity.

First, then, current flight control software was examined to determine its domain and the operations which were the critical cost drivers in the code. The domain of flight control software is defined as the language constructs used, the number and types of constants or variables employed, limits in the range of the variables and constants, and the control structure used in the software. The flight control software examined was the Integrated Flight and Fire Control Software, Ada Language (University of Colorado at Colorado Springs (UCCS), 1986) written for the F-15, the Advanced Fighter Technology Integration (AFTI)/F-16 Development and Integration Program, DFCS Phase Final Technical Report (Henley and others, 1984), and the software used in the

ABICS-II (Landy and others, 1986) project. This examination allowed the domain of flight control software to be determined and Ada code developed to test different aspects of this domain.

The ABICS-II software was chosen because it was the first, and to this date only, use of Ada code for actual implementation of flight control software (Voelcker, 1987: 44). The UCCS software was chosen because it offered another approach which could have been used, although it was never implemented, to program flight control software in Ada for the F-15. As the AFTI/F-16 design code (as the name implies) is in the forefront of integrating new technologies into current and future aircraft (Voelcker, 1987: 48), the software developed was based on the AFTI/F-16 flight control design. An analysis based on manually counting specific types of statements (add, multiply, assignment, etc.) provided a definition from which tests could be developed which would measure the ability of an implementation of Ada to implement flight control software effectively and efficiently.

In addition to developing tests, the ACEC, PIWG, and other available benchmark test suites were examined to determine if any of their tests were suitable for use in evaluating compilers for flight control applications (as defined by the previous computations). This examination was accomplished by comparing the constructs of Ada tested by the benchmarks and determining if the same constructs were present in a

a usage similar to their usage in flight control software. Any tests which contained Ada constructs in the same frequency and used those constructs in the same manner as flight control software, or tests which were able to be modified to do so, were incorporated into the research project test suite. Tests were designed for language features for which no suitable tests could be found which simulated the flight control domain.

All benchmark tests that could be run on a DEC VAX-11/780 or a DEC VAX-11/785 under the VMS operating system were assumed to execute properly, based on the facts that the DEC VMS Ada compilers are validated and that each benchmark had been previously executed on other machine/compiler combinations with no errors. Proper execution was defined as correct results being achieved for numeric calculations and the correct path chosen in conditional statements based on the conditions at the time the statement was executed. The tests were then executed on each of two compilers and two 1750A processors. The compilers and processors were chosen because they are representative of the seven validated 1750A Ada compilers and twelve (as far as is known) 1750A processors available (Siegert, 1987; Pitarys, 1987). Additionally, the chosen compilers and processors are currently being used in DOD evaluation tests, funded research, or the development of current and future projects (Siegert, 1987; Pitarys, 1987). Each test was run a number of times on the

same processor/compiler combination. Statistical analysis of the execution times was used to determine if there were any significant differences between runs on the same processor/compiler combination or between processor/compiler combinations. The results compared the compilers based on test execution speed and size of code generation.

No attempt was made to make a single rating of compilers based on all of the tests. Such a composite rating was not done because each benchmark typically measures a limited feature(s) within a language. The importance of a particular feature is dependent on the application for which it will be used. Each application manager will need to make the determination of what, if any, tradeoffs are acceptable, based on the test results, and choose the appropriate compiler/processor combination.

The research was finished when all of the benchmarks were run, either successfully or unsuccessfully, on each of the 1750A processor/compiler combinations and the results analyzed to determine the suitability of the tests in evaluating the various combinations for implementing flight control software. The success of the tests in measuring the ability of Ada to implement flight control software was validated by discussing the design and implementation specifics with experts in the fields of flight control and compiler benchmarking.

Research Gain

This research provided the first series of tests that allows flight control researchers to compare, based on the speed of execution and the code size generated, the effectiveness of various Ada compilers/RTS for flight control applications. Also, this was the first known attempt to identify and characterize the critical cost drivers in flight control software.

Thesis Organization

Chapter II presents work that has been done previously on Ada compiler evaluation and on flight control characterization. The domain of flight control applications is determined based on examples from the literature. The evaluation of Ada compilers for general applications and real-time systems is examined to understand the process of benchmarking and to look for tests which may be suitable for flight control.

Chapter III explains the system designed to evaluate Ada compilers for flight control applications. The method by which Ada constructs were chosen for evaluation and how the constructs were measured is explained.

Chapter IV provides a detailed explanation of each benchmark. The development of each benchmark is explained and the method used to measure the execution speed described.

Chapter V describes validation of the designed tests.

Chapter VI explains the conclusions obtained from the research. The problems encountered in the research effort, possible future research, and the benefits gained in the research are described.

II. Literature Review

To determine if Ada can effectively implement flight control software, the problem defined in Chapter 1, two areas must be addressed. Initially, the domain of flight control software must be defined. Consequently, this chapter surveys the existing sources of flight control software to characterize it. Secondly, the capability of Ada to produce efficient and effective code in this domain must be evaluated. So this chapter examines the existing research that has been conducted in Ada compiler evaluation for flight control-sensitive features.

Flight Control Software Domain

Recall from Chapter 1 that the domain of flight control software is defined as the flight control functions implemented, language constructs used, and algorithms employed in the software. The first step in determining the domain of flight control software was to determine which aircraft would be included in the domain definition. Calls to all the major contractors and the project offices for the X-29 and the advanced tactical fighter (ATF) aircraft revealed these projects were still in the design phase and could not provide data useable for the domain definition. Data was obtained for the ABICS-II F-15 aircraft (Landy and others, 1986), the IFFC F-15 flight control law study (UCCS, 1986), and the AFTI/F-16 project (Henley and others, 1984).

Examination of documents from the preceding studies produced the following functional area decomposition:

1. Executive. The executive is designed to control and schedule the execution of the other tasks in flight control software. The executive may include checks on certain interrupts to insure completion of tasks (Ramage and others, 1981:21-22; Landy and others, 1986:4-2 to 4-5).

2. Redundancy Management or Selection/Monitoring of Inputs/Outputs. This portion of the software accepts inputs from aircraft sensors and checks them for validity by comparison to other sensor inputs and for reasonableness based on predefined limits. The inputs are forwarded to the computational phase upon acceptance. The outputs follow the same validity checking but flow from the computational phase to the control surfaces of the aircraft (Ramage and others, 1981: 22-24; Landy and others, 1986:4-8 to 4-9).

3. Control Laws. The mathematical calculations which solve flight dynamics equations are computed in this portion of the software. The inputs from sensors, pilot selections, and possibly previous inputs or outputs are used to arrive at commands for aircraft control surfaces (Ramage and others, 1981: 24-25; Landy and others, 1986:4-5 to 4-8).

4. Pilot Interface. This portion of the software handles the input and output of data to displays in the aircraft cockpit.

5. Others. Additional modules of the flight control software include system tests, startup and restart routines, and failure management modules.

An examination of the time and space used by each of these functional areas determines which functional areas are critical time and space cost drivers. Table II provides a sample analysis of such functional area resource usage.

The 0.0 percentages recorded in time usage reflect the fact that the system monitor and startup/restart functions are not active during flight but operate only during ground operations. The spare time recorded reflects time which is used to execute non-critical functions. The spare memory recorded reflects unused memory locations. The figures in Table II are for the AFTI/F-16 aircraft: the figures for the ABICS-II F-15 aircraft are significantly different, although the distribution of resources among the functional areas is similar with control law and redundancy management functions the critical cost drivers, using 29 and 31 percent of the time and 13 and 15 percent of the memory respectively (Landy and others, 1986:4-16, 8-6).

Table II. Functional Area Resource Utilization (Yousey and others, 1984:3-2)

Functional Area	Time	Memory (percent)
Executive	6.0	2.7
Redundancy Management	43.6	23.3
Control Laws	30.0	34.0
Pilot Interface	12.0	5.4
System Monitor	0.0	17.7
Startup/Restart	0.0	2.2
Spare	8.4	15.7
Total	100.0	100.0

The next step in defining the domain of flight control software was identifying from the three projects/studies previously mentioned the Ada language constructs used to implement flight control functions.

The ABICS-II study identifies key Ada constructs pertinent to flight control (see Table I, page 3) and the interaction of these constructs based on actual experience in using Ada to implement flight control software for the F-15, the first such use of Ada. Floating point arithmetic was used for computing the control laws and was implemented in software rather than hardware (Landy and others, 1986: 9-10 to 9-11). The executive, control laws, and a built-in test were coded in Ada (Landy and others, 1986:2-2). Interrupt handling, input/output (I/O), some logic and math functions, and redundancy management were coded in assembly language (Sargent, 1987). Iterative control, conditional control, tasking, exceptions, and the capability to interface with various hardware components were stressed as important Ada constructs in flight control applications (Landy and others, 1986:9-3 to 9-14). Additional Ada constructs found to be useful were packages, functions, and procedures: these additional Ada constructs allowed programs to be decomposed into modules, speeded development, and allowed quick adapting of changes because of the isolation of the effect of a change to a small module set (Landy and others, 1986:9-3 to 9-5).

The IFFC Software (UCCS, 1986) provides an alternative method of implementing flight control software in Ada for the F-15. The IFFC software has never been used in flight and includes no assembly language routines. The IFFC software contains no exception handling or tasking, nor does it interface with the underlying hardware. Otherwise, it contains the same Ada constructs used in similar frequencies as the ABICS-II software (UCCS, 1986).

The AFTI/F-16 report (Henley and others, 1984) contains the software design of the AFTI/F-16 digital flight control system. The design documentation includes psuedocode and detailed control law computation algorithms. Using fixed point arithmetic for control law computation (Ramage and others, 1981:5-6), the AFTI/F-16 flight control software was coded in assembly language. However, the pseduo code allows the algorithms to be analyzed in a higher order language. The psuedocode and control law computations algorithms reveal that the same types of Ada constructs used in similar frequencies are needed for the AFTI/F-16 as are needed for the ABICS-II and IFFC flight control implementations (Henley and others, 1986).

The final step in defining the domain of flight control software is to determine the algorithms used to implement flight control software. An examination of the three flight control software documents/studies previously listed show that the specific algorithms used vary from aircraft to

aircraft for flight control software. The differing algorithms are caused by the varying number and type of control surfaces, sensors, and the redundancy built into the hardware of the aircraft. However, the various algorithms include the same types of operations in similar frequencies which include evaluating inputs and outputs for consistency, computing similar flight dynamics equations, and interpolating between data points (Yousey and others, 1984:4-14, 3-64; Henley and others, 1984:13-1 to 13-192; Landy and others, 1986:4-7 to 4-9).

Benchmark Testing

The evaluation of compilers through the use of benchmark tests has been used often in the past (Bennell, 1975: vii). Although, the use of benchmarks for real-time systems using Ada has only recently been started, numerous benchmark programs exist in the public domain for Ada. The proto type Ada Compiler Evaluation Capability (ACEC) (Hook and others, 1985) is a series of benchmark tests, each of which evaluates an individual Ada language feature or a combination of Ada language features. The prototype ACEC, developed for supermini or larger computers from tests available in the public domain (Hook and others, 1985:1), also includes a support system which allows timing statistics to be gathered.

The tests in the prototype ACEC are divided into two main categories. The *normative* category contains tests of

features required to be part of the Ada language by the LRM. The *optional* category includes tests of Ada constructs not required by the LRM. Both categories contain tests of some of the Ada constructs pertinent to flight control as described in the previous section.

Some ACEC tests rely on a control and a test version of each program. The cost of implementing a particular Ada feature is determined by the difference in the time required to execute the two differing versions of the test. The *synthetic* tests in the prototype ACEC rely on only a single test which is executed on two systems, with the difference in execution speed used for comparison of the two systems.

The Boeing Corporation is currently under contract to the Department of Defense (DOD) to develop the full ACEC, thus providing definitive tests across a wide spectrum of software applications, including real-time avionics for collection and analysis of Ada compiler performance data (Ada Information Clearinghouse (AdaIC), 1987).

The PIWG benchmarks (Squire, 1986) are similar to the ACEC benchmarks. Some tests are included in both test suites. The PIWG test suite includes more tests evaluating Ada constructs for program structure and control. No tests of simple Ada constructs (individual math functions, assignments, etc.) are included. The PIWG test suite has a number of tests which evaluate the same language feature under different conditions. Most tests are synthetic tests. Included

within the benchmarks are (1) routines which thwart the compiler's attempts to optimize away the feature(s) to be measured, and (2) a dual looping structure which allows the time needed to record timing measurements to be ignored. The PIWG benchmarks were also designed for use on supermini or larger computers, but they also include some programs which are intended to allow the benchmarks to run on 1750A machines. The PIWG benchmarks also include an elaborate support system for obtaining test results.

The Air Force Wright Aeronautical Laboratories (AFWAL) Avionics Laboratory is currently running the PIWG benchmarks on 1750A architectures which are either being used in DOD funded research or have been used in avionics applications (Pitarys, 1987). Additionally, they are attempting to develop benchmarks specifically oriented to avionics applications. This work is progressing slowly due to problems with data conversion of timing information and with the lengthy amount of time required to load the benchmarks into the 1750A machines (Pitarys, 1987). For example, it currently takes from 25-45 minutes to compile, link, load, and run a single benchmark on one 1750A processor, and there is a high demand for available processors.

During his thesis research, Witt uncovered a number of issues which were pertinent to real-time avionics systems using Ada (Witt, 1985:28-41). The issues he raised pertinent to flight control software are tasking considerations and

exception handling. He executed a number of the prototype ACEC benchmarks on a DEC VAX-11/780 and a Data General computer to provide insight into the cost of using Ada to implement real-time avionics. Witt concluded that, since many real-time embedded systems issues are not adequately addressed in the prototype ACEC, developers should produce benchmarks for features critical to their applications and contribute them to the public domain to benefit the whole Ada community.

Craine, continuing the research begun by Witt, developed a series of benchmarks which tested the overhead associated with task rendezvous, run-time constraint checking, using many tasks or many *select* statements within tasks, reordering of *select* statements, non-invoked exceptions, compiler optimize options, pragma suppress, length clauses, and unchecked deallocation (Craine, 1986:44-65). Craine also concluded that further examination of avionics software was needed to construct more representative benchmarks for avionics.

Much of the previously described work used superminicomputers, or larger computers, as target machines. Flight control software, however, is executed on 1750A-class architectures. Thus, many existing results on Ada compiler performance are not necessarily transferable to flight control applications. The only complete benchmark testing which has been done on the 1750A architectures has been done by the contractors producing or using those architectures (Mellby,

1987; Startzman, 1987). But such testing is proprietary and thus is unavailable for general use.

III. System Design

This chapter explains the design of a system, developed based on the existing partial solutions described in Chapter II, for addressing the thesis problem of evaluating Ada compilers for flight control software based on the speed of execution and the size of the code generated. Evaluation system requirements, alternative evaluation systems, and the the design of a system, adapted from the alternatives, to meet these requirements are described.

Compiler/RTS Evaluation Requirements

Speed of execution is crucial in flight control applications. If a compiler/RTS and the processor it is hosted on do not meet a system's real-time or memory space constraints, the compiler/RTS and processor combination is unsuited for that application (Westermeier and Hansen, 1985: 602; Lahn and others, undated:1-2). So a system used to evaluate compiler/RTS and processor combinations must capture the execution time of the produced code and a means must be available to determine the memory space required. Next, to confirm elimination of transient system (both hardware and software) effects, the results of the system tests must be repeatable. Further, the system must be able to isolate the construct(s) being measured from other language constructs used in the system which may influence the results of the system tests. Finally, the validity of the system requires

that the tests truly represent the flight control domain in both the number and types of language constructs used and in the manner in which the constructs are used. Also, any constraints on values must be incorporated. For example, if the value of a variable may range from +25.0 to -3.0 then the system must insure that values outside that range are not allowed.

Alternative Evaluation Systems

The three basic methods for evaluating compiler performance are application domain programs, testing a complete application; composite benchmarks, testing integrated features; and singular benchmarks, testing an individual feature (Clapp and others, 1986:760; Craine,1986:13).

To embed timing routines within actual flight control application programs would insure that the timing evaluation reflected the true nature of flight control software.

> This is an established reliable method for quantitatively assessing overall time and space performance characteristics of the generated object code when the intended application domain is well understood.
>
> (Bassman and others, 1985:151)

However, recall from Chapter II that flight control software is currently written mostly in various assembly languages and/or JOVIAL. Running application programs in these languages would not illuminate Ada's suitability for flight control software. For this reason, current application domain programs were not used.

Composite benchmarks (or *synthetic benchmarks*) may also be used to evaluate the speed of execution of code generated by a compiler. In composite benchmarks, a number of language features are incorporated into one test program. The language features in composite benchmarks are executed with the same frequency as they appear in the actual application domain. The use of composite benchmarks thus requires a sample of the application domain software or detailed design documents to be examined to determine the frequency of use of language features. The AFTI/F-16 design documents contain very detailed pseudocode and execution sequence charts which contain all the information needed to construct composite benchmarks. Because flight control software includes a rich mixture of language constructs (Landy and others, 1986:9-1 to 9-15; Ramage and others, 1981:21-26), composite benchmarks, modeling interaction of constructs, more accurately reflect flight control software than singular benchmarks. The composite benchmarks developed for this research are very similar to application domain benchmarks because they were derived from application domain designs and code.

The final approach considered was the use of *singular benchmarks*, which evaluate specific language features. This method requires isolating each feature to be tested, achieving accurate and repeatable results, and eliminating the effect of the computer system (other than the compiler/RTS) on the test (Clapp and others, 1986:760-761). For singular

benchmarks to be useful, isolations of features in an application, and a time breakdown of feature use within the application are needed. As an example of why such isolation of a single feature is not feasible for flight control software, consider the following. The calculation of a value for an output to a control surface is based on the control surface's last position and the current aircraft speed, altitude, and direction. With different values for the speed, altitude, and direction inputs, the calculation involves varying numbers of mathematical operations. Thus the time required to obtain the output value depends on not only the number of mathematical operations but also the time needed to determine the preconditions of the calculation. Flight control software is *filled with such interdependencies* (Landy and others, 1986:9-1 to 9-15; Ramage and others, 1981:21-26). Consequently, singular benchmarks were not used.

System Overview

The system designed to solve the thesis problem statement required tests be designed which evaluate Ada's ability to implement flight control software.

Flight control software must first, therefore, be characterized before a system can be designed which will examine the effects of using Ada for flight control software. The problem with characterizing flight control software in a generic fashion is that each aircraft's flight control

surfaces may differ in both number, size, and location. Thus, the control law computations will be less or more complicated which leads to higher or lower time and space requirements. Also, computer resources and the way in which these resources are used vary between aircraft. Some systems dedicate separate computers to compute control laws for different axes (Landy and others, 1986:3-1) while other systems use all computers for all axes (Ramage and others, 1981:4-7). For example, the algorithm to compare two values for consistency is much simpler than the algorithms to compare three values, taken two at a time.

The AFTI/F-16 is representative of current fighter aircraft because: (1) it is slightly unstable which provides the ability to increase performance, (2) it is a fly-by-wire system with integrated avionics applications, and (3) its redundancy management scheme isolates and adapts to system failures (Barfield, 1987; Toolean, 1987; Johnston, 1987; DeThomas, 1987). The overall design of the AFTI/F-16 is representative of the current trend of redundant, fly-by-wire, slightly unstable aircraft (Joslin, 1987; Toolean, 1987). For the preceding reasons all software developed to determine Ada's ability to implement flight control software used the system design and algorithms for the AFTI/F-16 aircraft.

Next, with flight control software having been characterized, the functions within flight control software which

are the critical cost drivers must be determined so pertinent tests can be developed. There are two critical factors in flight control software, execution time and code size. As shown in Chapter II, the AFTI/F-16 has five main functions which are used in flight: the executive, control law computation, selection/monitoring of inputs/outputs, failure management, and pilot interface. These functions were examined to determine the language constructs which they use. Manually converting the diagrams and algorithms in the design documents for the AFTI/F-16 to Ada provided the domain definition of flight control software. The conversion from flight control system design to Ada involved implementing limits on values and conditional inputs with conditional statements, incorporating all mathematical computations, and initializing all constants to the values used in the design. All benchmark tests designed were based on this definition.

Benchmark Tests

The following sections explain which flight control functions were analyzed and the method by which tests simulating these routines were constructed.

Control Laws. As defined in Chapter II, flight control law computations are mathematical solutions to flight dynamics equations. These computations are needed to keep the aircraft stable during flight. Also recall from Chapter II that in one typical application 30 percent of the time and

34 percent of memory is devoted to control law computations; thus, the control law function is a critical cost driver in flight control software. Conditional statements which determine the state of the aircraft and arithmetic operations and assignment statements based on these conditions are the Ada constructs used in this functional module. Because the conditions evaluated by the conditional statements control which arithmetic operations will be executed, both the conditional statements and arithmetic statements are the Ada constructs which drive the cost of control law computation.

The AFTI/F-16 has four modes - standard normal, air-to-air gunnery, air-to-surface gunnery, and bombing (Yousey and others, 1984:3-68 to 3-69). Each mode tailors aircraft performance to enhance handling or stability for a particular mission, as described by the mode names. The standard normal mode, from which all the other modes were derived, was chosen for development of the benchmark because it is representative of all control law computations (Barfield, 1987; Johnston, 1987; DeThomas, 1987). The benchmark reflected the frequency of the Ada constructs identified as the cost drivers for control laws. Because the benchmark is an Ada implementation of actual flight control software it will provide a good indication of whether one compiler/RTS and processor is more suitable than another combination for flight control law computations in terms of the speed of execution and the memory needed to implement the benchmark.

Gain Table Interpolation. A separate element in flight control law computation is the computation of gains, limits, and coefficients (Henley and others, 1984:13-26 to 13-30). The values of these parameters are determined based on various flight data such as angle-of-attack, air pressure, and altitude. The values are determined by interpolating between data points which are in tables (arrays) in the flight control computer. The criticality of these computations is based on the same reasoning used for the control law computations. The time and space criticality of the gain values is presumed because they are needed for the control law computations (Henley and others, 1984:13-26).

A compensation filter frequency gain was selected for benchmark testing because it represented a typical gain table interpolation (Barfield, 1987; Joslin, 1987). The benchmark embodies a method in which all gain values could be computed. The benchmark is an Ada implementation of a method to compute double interpolations of actual flight control data and thus will provide a good indication whether a compiler/RTS and processor is more suitable than another combination for flight control law computations in terms of the speed of execution and the memory needed to implement the benchmark.

Redundancy Management. Redundancy management is a collective term which encompasses selector/monitor and failure management functions. These functions are required to select

and monitor sensor inputs and the outputs to control surface actuators. Failure management is invoked upon determining that a failure has occurred. Redundancy management is necessary to insure that the aircraft does not erroneously act upon incorrect commands given by failed hardware or software. In addition to their importance in safety, these functions are critical because, in one typical application, they use 43.6 percent of the time and 23.3 percent of the space used by flight control software (Table II, page 14). The Ada constructs used in redundancy management are conditional statements, mathematical operations, and procedural calls.

The developed benchmark is an Ada implementation of the algorithm used to monitor the leading edge flap (LEF). The LEF monitor function was chosen because it involves all aspects of redundancy management: monitoring input values, checking for inconsistencies between different input values and the same input values from different computers, checking inputs and outputs for consistency, and calling failure management when conditions warrant it.

Test Selection

The functional areas for which tests were developed are the areas which required the most time and memory space in one typical flight control software implementation. The other areas were not chosen for benchmark development because they either are not used during flight or are in use a much

smaller percentage of in-flight time and use much less space as compared to the chosen functional areas. The benchmarked functional areas use 80.3 percent of the in-flight time and 67.9 percent of the memory space used for flight control software.

IV. Detailed Design

This chapter explains the steps taken in developing and executing the software necessary to measure the time needed to execute Ada constructs identified in Chapter II as the critical cost drivers in flight control applications. Each test description presents the function of the test and explains the process used to evaluate the tested function. In Chapter III the selection and importance of the tested functions as the critical cost drivers for flight control software was explained. The following sections explain in more detail how the testing was conducted and why. The sub-headings are the same as the sub-headings in Chapter III.

Control Laws

Function. The function of this test is to measure the speed of execution of the control law computations found in flight control software. Control law operations are critical for two reasons, as explained in Chapter II; they consume 30 percent of the time that flight control software is executing, and they are necessary to keep the aircraft in stable flight.

Description. Initially, the documents from the IFFC (UCCS, 1986), ABICS-II (Landy and others, 1986), and the AFTI/F-16 (Henley and others, 1984) projects were examined and the frequency distribution of language constructs determined by manually counting the constructs and noting the

types of operands and operations, where appropriate. The two F-15 software products used floating point calculations while the AFTI/F-16 software used fixed point calculations. Also, the different number of flight control computers, the manner in which they are used, and the different control law computations used for each aircraft made it impractical to make one benchmark representative of both aircraft. A single benchmark could not be developed because the number of language constructs required to implement the control law computations varied from one aircraft to the other by over 50 percent of the total for the two aircraft.

The next step was to decide whether to make dual benchmarks, one for each aircraft, or to concentrate on one aircraft. Recall from Chapter III that the AFTI/F-16 was chosen because it more closely represented present and near future fighter aircraft. Table III contains the percentage of language constructs and operands used in the AFTI/F-16 control law computations. The benchmark developed to evaluate control law computation is an Ada implementation of AFTI/F-16 control laws and contains the same percentage of constructs and operands as shown in Table III (in fact, the benchmark contains the same number of constructs and operands). The software design documentation for the AFTI/F-16, specifically the algorithms and values in Chapter 13 and Appendices B, C, and E, provided the definition of a typical flight control software design (Henley and others, 1984). The benchmark

Table III. Language Constructs and Operands Used in AFTI/F-16 Flight Control Law Computations

Multiplications	- 32 %
Divisions	- 6 %
Additions	- 17 %
Subtractions	- 6 %
Assignments	- 20 %
Absolute Values	- 2 %
Conditionals	- 17 %
Variables	- 70 %
Constants	- 30 %

implements the standard normal mode control laws for the AFTI/F-16. During development, it became clear that not only the compilers and processors but also the manner in which the code was written could have a noticeable effect on the execution speed of an application. Because speed of execution is so critical to flight control software, this benchmark was coded in two styles: a modular, information-hiding style where variables and constants were declared and used in localized modules, and a global style where all variables and constants were declared in the main procedure and thus, usable in all modules. Appendix A has examples of the two coding styles used. The benchmark uses a set of data designed to follow a normal path through the control

laws. A normal path is defined as the path where none of the structural or performance limits of the aircraft is approached. A normal path was chosen because it represents the normal operating range of the aircraft (Hicks, 1987; Barfield, 1987). Thus the benchmark, as designed, is only completely suitable for evaluating Ada compilers/1750A processsors for the AFTI/F-16 in standard normal mode during a normal flight path; however, these limitations are mitigated by the opinions expressed in Chapter III and previously in this chapter that the AFTI/F-16 is representative of current fighter aircraft and the non-standard normal modes of operation are only slight variations on the standard normal mode.

Gain Table Interpolation

Function. The function of this test is to measure the speed of execution of accessing data in a gain table and interpolation of a value between data points. Gain table access and interpolation are critical because if gain table calculations are too slow the control laws will not have valid data for computation and aircraft performance and stability could be adversely affected.

Description. The values to be used in this benchmark were extracted from Appendix E and Chapter 13 of the software mechanization for the AFTI/F-16 (Henley and others, 1984). Appendix B of this thesis contains a sample of the

double interpolation method used between four data points in the benchmark designed to represent this function (Barfield, 1987). The benchmark contains a representative gain table and interpolation. One sample was considered sufficient because the other gain tables and their interpolations use the same methods (Barfield, 1987). An estimate of the time needed to compute additional gain values can be obtained by multiplying the benchmark result by the desired number.

Redundancy Management

Recall from Chapters II and III that redundancy management is used to monitor inputs/outputs, make comparisons of the data received, and act on the results of those comparisons. One benchmark was developed for this function. The leading-edge flap (LEF) monitor was chosen because it includes all the operations of redundancy management as stated above. A single benchmark is suitable because it is representative of the complete redundancy management area.

The LEF monitor is a series of conditional statements with a few mathematical calculations and a call to the failure management procedure when needed (Henley and others, 1984: Chapter 6). The algorithms for the LEF monitor were coded in Ada. The code was initially designed not to call the failure management routine because it is not a critical cost driver, as described in Chapters II and III, however; this meant that the effect of detecting and operating with

failures was not evaluated for the LEF monitor algorithm. To test operation under these conditions, a second version of the benchmark was developed which included detecting failures and operating the algorithm under failure conditions. The second version provided a means to gauge how the benchmark operates under differing sets of conditions. Because the benchmark was developed from the same algorithms as used in the actual LEF monitor for the AFTI/F-16, the results provide a good comparative evaluation of varying processor/compiler combinations' abilities to implement redundancy management software.

Test Procedures

A support system adapted from the PIWG benchmark support package was used to obtain timing data and output the timing data to a monitor for recording. To determine if support software would affect the results, two different test programs were run with the support software and without it. Because there was no difference for either program in the times recorded, it was assumed the support software did not affect timing measurements.

To eliminate any effects obtaining timing measurements might have on timing data the dual loop benchmark method was used. In this method, a *control loop* is used to eliminate the effects of calling a timer function. The control loop contains only enough code (generally a call to a remote

procedure) to keep it from being eliminated by the compiler. The *test loop* contains the same code as the control loop and the feature(s) to be measured. The difference in the time required for the control loop and the time required for the test loop is the time required to execute the feature(s) being measured. Figure 1 contains an outline for this type of benchmark. To insure the timing measurement is greater than the smallest time the system clock can record the loop is executed until the measured time is greater than the clock resolution. The time to execute the benchmark is determined by dividing the number of iterations into the elapsed time.

```
Loop
                Control_start := Timer;
                   Loop for 1 to Count
                      Remote();
                   End Loop;
                Control_stop := Timer;

                 Test_start := Timer;
                    Loop for 1 to Count
                       Remote();
                       Feature_to_be_measured;
                    End Loop;
                 Test_stop := Timer;

      Feature_Time := (Test_stop - Test_start)
                      - (Control_stop - Control_start)

      If Feature_Time > minimum system time then
         Exit;
      Else
         Count := Count + 1;
End loop;
```

Figure 1. Dual Loop Benchmark

On multi-user systems, timing data is often invalidated by the operating system sharing time among users and peripheral devices such as printers and disk drives depending upon the demands placed on the system. Thus, widely varying times are possible for the same benchmark. Because 1750A processors are single-user devices with no peripheral devices it was assumed variance would not be a problem. To test this assumption the test programs discussed earlier in this chapter were run five times on the 1750A architecture: the times were identical for each run. On multi-user systems, variances always appeared among five or fewer runs.

To obtain timing data, the PIWG benchmarks for 1750A processors use the Ada CALENDAR package, which measures wall-clock time and not CPU time. To determine if the PIWG method was accurate an assembly language routine was developed, in collaboration with Joyce and Pitarys, which used a real-time clock, bypassing the Ada package (Joyce, 1987; Pitarys, 1987). The same sample tests were run using both methods to record times. Assembly language routines were used in the research because the resolution of the real-time clock was finer than the resolution in the Ada calendar package and produced more accurate timing data.

V. Test Analysis

This chapter evaluates the validity of the benchmark tests for solving the thesis problem: to show the comparative capability of an Ada compiler/RTS and 1750A processor combination to effectively and efficiently implement flight control software. This capability is judged upon the ability of a combination to implement the flight control-critical Ada constructs and flight control algorithms described in Chapters III and IV. Detailed results of the developed benchmark tests run on two 1750A processors using two Ada compilers is at Appendix C. The compilers and processors tested in this research are labeled generically throughout the results to permit the widest distribution of the thesis. Machine-readable versions of the benchmarks may be obtained from the Air Force Institute of Technology[1].

Validation Method

Validation that a benchmark provides a means to judge comparative capability of 1750A processor and Ada compiler combinations is based on the opinions of personnel knowledgeable in aircraft flight controls and flight control software. Validation is handled separately for each benchmark.

---

[1] Department of Mathematics and Computer Science, WPAFB OH 45433, (513) 255 - 3098.

Control Laws

The control law computation benchmark is suitable for evaluating Ada's ability to implement control law computations. The benchmark provides an indication about the time and space needed to implement a typical aircraft's flight control software in Ada. The performance during atypical conditions and the way the flight control software handles these conditions should be considered in order to obtain a better comparison (DeThomas, 1987).

The control law computation benchmark is valid, to a degree, across aircraft because, in general, the same types of aerodynamic computations are needed to keep any aircraft stable. The degree of validity will depend on the similarity between the aircraft for which an evaluation of compilers is to be conducted and the AFTI/F-16. The benchmark would be a better evaluation if it included a framework to alter data dynamically. For example, the benchmark would be improved if it included a complete scenario from take-off to landing the aircraft (Barfield, 1987).

The control law benchmark does give an indication of the comparative capabilities of Ada compiler/RTS and 1750A processor combinations to operate in the standard normal mode of operation. A better indication could be achieved by encoding more of the modes of operation or computing more of the variable data rather than assigning set values to the variables (Hicks, 1987).

The control law benchmark gives an indication of how well an Ada compiler/RTS and processor computes control laws. Increasing the number of paths taken through the control laws to test the effect of different conditions would be a suitable extension of the benchmark. For example, widening the range of the data inputs (which drives the path the computations take) would provide a more robust evaluation of the compiler's ability to keep the aircraft stable (Lewantowicz, 1987).

Gain Table Interpolation

The gain table interpolation benchmark provides useful data to compare compilers for flight control software because (1) they calculate gains accurately; and (2) the calculation of gains is a critical portion of flight control software. Encoding multiple gain tables would provide a better indication of Ada's suitability, especially in generated code size (Barfield, 1987).

The gains table interpolation benchmark is useful in determining code execution speed and size, because gains for are calculated in basically the same way for different types of aircraft. The gain table interpolation could be enhanced by including different types of interpolation. For example, inclusion of a gain table which has fewer data points or a gain table which is entered with only one input (Joslin, 1987).

Redundancy Management

The redundancy management scheme of the AFTI/F-16 provides an indication of Ada's suitability for use across aircraft because the AFTI/F-16 design is being used to design redundancy management schemes for other aircraft. The LEF monitor portion of the AFTI/F-16 design does implement all of the major elements of the redundancy management scheme and is representative of the overall scheme (Hicks, 1987).

Although it may not encompass the complete range of redundancy management software, the benchmark provides a partial method for evaluating Ada's ability to implement another element of flight control software (Barfield, 1987).

The LEF monitor benchmark provides one point at which to judge a compiler's ability to implement flight control software. The test should be expanded to include more types of conditions (i.e., additional mixtures of failures) to see what, if any interdependencies exist and determine their effect on execution speed (DeThomas, 1987).

The LEF monitor provides a good indication of a 1750A processor and compiler combination's ability to execute redundancy management schemes because it incorporates conditional execution, procedural calls, and a mathematical model which are the major functions of any redundancy management scheme. Including more of the redundancy management scheme in the benchmark would enhance the value of both the execution speed and code size measurements (Joslin, 1987).

Summary

The developed benchmarks, based on the expert evaluations in the previous sections, predict the utility of compiler/RTS and compiler combinations for flight control software with the following limitations:

1. Any aircraft which substantially differs from the AFTI/F-16 in implementation of flight control software will receive results which may not be transferable to that aircraft. The judgement of whether an aircraft is substantially different is subjective in nature and thus is left to the individual user to determine.

2. The benchmarks were designed using the standard normal mode of operation and well within the performance limits of the aircraft. Judgements on utility at the limits of the aircraft can easily be implemented by changing the variables in the benchmark. For other than standard normal mode, additional modules would need to be coded.

VI. Conclusions and Recommendations

This chapter provides an explanation of how well the thesis objectives were achieved, conclusions reached based on the results of the thesis research, additional observations which were uncovered during the research, and recommendations for further research. The primary objective of this research was to develop tests which enabled different Ada compiler/RTS and 1750A processor combinations to be compared in execution speed and code generation size for flight control software suitability. Two additional objectives were needed to meet the primary objective: (1) to characterize flight control software so suitable tests could be developed, and (2) to run the developed tests on compiler/processor combinations representative of current processors being used in DOD aircraft or research to insure the tests would run in the target environment.

Achievement of Objectives

The characterization of flight control software involved examining current software for general similarities. The examination showed that a general solution was not entirely realizable so the current trend in fighter aircraft was determined and the characterization made to fit this trend. The AFTI/F-16 aircraft was chosen as the general characterization of high-performance fighter aircraft.

The development of benchmarks involved determining tne Ada constructs which are the critical cost drivers in the previous characterization and developing programs (the benchmarks) which were representative of the use of these constructs in the application domain. This was done by examining the functions of flight control software which use the most time and space, selecting them as the critical routines, examining these routines to determine the Ada constructs used in them, and writing benchmarks to test the cost of implementing the routines/Ada constructs. Three benchmarks were developed along with support software used to measure execution time and isolate the features to be tested. This objective was partially successful in that portions of the flight control functions which are the critical cost drivers were encoded in Ada and the suitability of the developed benchmarks was endorsed by experts in flight controls and flight control software, as described in Chapter V.

The developed benchmarks were executed on two 1750A processors using two compilers. The compilers and processors used were chosen based on their use in DOD-sponsored research and development. The benchmarks were successfully compiled and linked on both compilers for both processors. When the compiled code was executed only one processor ran both compiled versions of the benchmarks; the other processor ran no programs from one compiler and only the smaller

benchmarks from the other compiler. This objective was partially achieved because the one processor did not run all benchmarks. Based on the findings of this research and concurrent research efforts it was concluded that the processor was at fault (Pitarys, 1987).

Conclusions

The developed benchmarks are valid for aircraft which have flight control software implemented similar to the AFTI/F-16; any aircraft which differs significantly in any one of the flight control functional areas from the AFTI/F-16 will obtain results less representative of its application domain. As described in Chapter V the difference between aircraft, and thus the degree of suitability of the benchmarks, is left to the individual(s) wishing to use the benchmarks. In general, the opinions of experts cited in Chapter V describe the benchmarks as representative of current fighter aircraft.

Accurate and repeatable measurements are achievable if care is taken to eliminate system effects and the 1750A's real-time clocks are interfaced to the timing measurement software.

Additional Observations

A compiler used for flight control software should include ADDRESS CLAUSES because they allow access to the underlying hardware. ADDRESS CLAUSES allow designers to

have direct access to inputs and outputs which are allocated to specific memory locations which mitigates the need for routines to get and send data. Additionally, a programmer will know that the data is always at the specified location with no need to initialize the data in software or insure the run-time system has not overwritten or optimized out the data.

Particular attention should be paid to compiler limitations. Any limits may affect the manner in which code is developed. For example, if only 100 external names can be referenced in a procedure, then many more local variables will be used, which can affect execution speed (as described in Appendix C). Similarly, the total number of variables allowed, limits on the number of units allowed to be WITHed, and nesting limits should be examined to insure a particular application does not exceed the limits of the compiler.

Recommendations for Further Research

Four areas are indicated for further research: (1) expand the scope of the benchmarks, (2) integrate the underlying hardware into the benchmarks, (3) develop a driver which simulates a series of inputs to the system, (4) obtain assembly language/JOVIAL code of flight control software for comparison to an Ada implementation.

The first area would involve incorporating more of the functional modules of the AFTI/F-16 into benchmarks and

running them on 1750A processors. This could include expanding the current benchmarks or developing new, separate benchmarks. This area would increase the accuracy of the benchmarks by making them more representative of the whole of flight control software.

The second area would involve integrating hardware into the benchmarks to provide more realistic input and output of data. This would increase the realism of the overall flight control system.

The third area would involve developing a driver which feeds data to the benchmarks and causes the benchmarks to take a number of paths through their algorithms. This area would increase the accuracy of the benchmarks by including a more representative sample of flight. For example, a driver could include take-off, flight, and landing conditions.

The fourth area would involve obtaining some actual flight control software and implementing the same algorithms in Ada, running both programs, and evaluating the results. This area would provide a sample of what the actual costs are of implementing flight control software in Ada as compared another language (rather than among Ada implementations).

Appendix A: Two Styles of Coding Control Laws

Two coding styles were used in developing the control law tests for flight control software. The two methods were different in the way in which variables and constants were declared and used. The global method allows all modules to use all variables and constants. Figure 2 contains a sample of the global coding method. As indicated by the scope line along the left side of Figure 2, variables A, B, C, D may be used throughout the MAIN procedure. Figure 3 contains a sample of the localized coding method. The scope lines indicate that variables A and B may be used throughout the MAIN procedure, while variable C can only be used within Module_1, and variable D can only be used within Module_2.

Scope of A,B,C,D

```
procedure MAIN
   A: integer := 0;
   B: integer := 1;
   C: integer := 2;
   D: integer := 3;
 begin

   Module_1 :
     begin
      A:= B + C;
      B:= A + C;
   end Module_1;

   Module_2 :
     begin
      A:= B + D;
      D:= A + B;
   end Module_2;
 end MAIN;
```

Figure 2. Global Coding Style

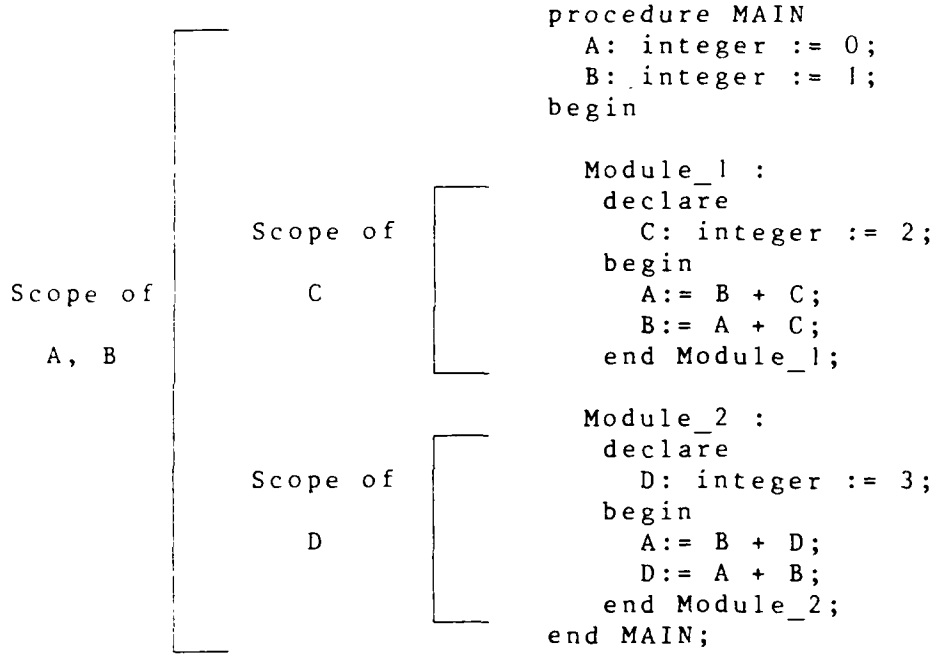


Figure 3. Local Coding Style

Appendix B: Gain Table Interpolation

Figure 4 contains a part of the gain table for a compensation filter frequency from the AFTI/F-16. The table is entered with two known values, speed (mach), and altitude static air pressure to sea level air pressure ratio (Ps/Po). The compensation filter frequency value is determined by a double interpolation of data points in the table. The dashed lines in Figure 3 graphically portray the following example. Enter the table with the known values 0.7 mach and 0.5 Ps/Po. The values for the compensation at 0.6 and 0.9 mach at 0.297 and 1.0 Ps/Po are determined because they are the values between which the known values fall. Let F(X,Y) represent the compensation at X mach and Y Ps/Po, the values from the table are:

$$F(0.6,\ 1.0) = 1364 \quad (4)$$
$$F(0.9,\ 1.0) = 1680 \quad (5)$$
$$F(0.6,\ 0.297) = 1154 \quad (6)$$
$$F(0.9,\ 0.297) = 1323 \quad (7)$$

To determine F(0.7, 0.5) the values of F(0.7, 1.0) and F(0.7, 0.297) are determined.

$$F(0.7,\ 1.0) = 1680 - ((1680 - 1364) * ((0.9 - 0.7) / (0.9 - 0.6))) \quad (8)$$

$$F(0.7,\ 1.0) = 1469.3 \quad (9)$$

$$F(0.7,\ 0.297) = 1323 - ((1323 - 1154) * ((0.9 - 0.7) / (0.9 - 0.6))) \quad (10)$$

$$F(0.7,\ 0.297) = 1210.3 \quad (11)$$

Next, the value of F(0.7, 0.5) is determined.

$$F(0.7, 0.5) = 1323 - ((1323 - 1210.3) * ((1.0 - 0.5) / (1.0 - 0.297))) \quad (12)$$

$$F(0.7, 0.5) = 1242.8 \quad (13)$$

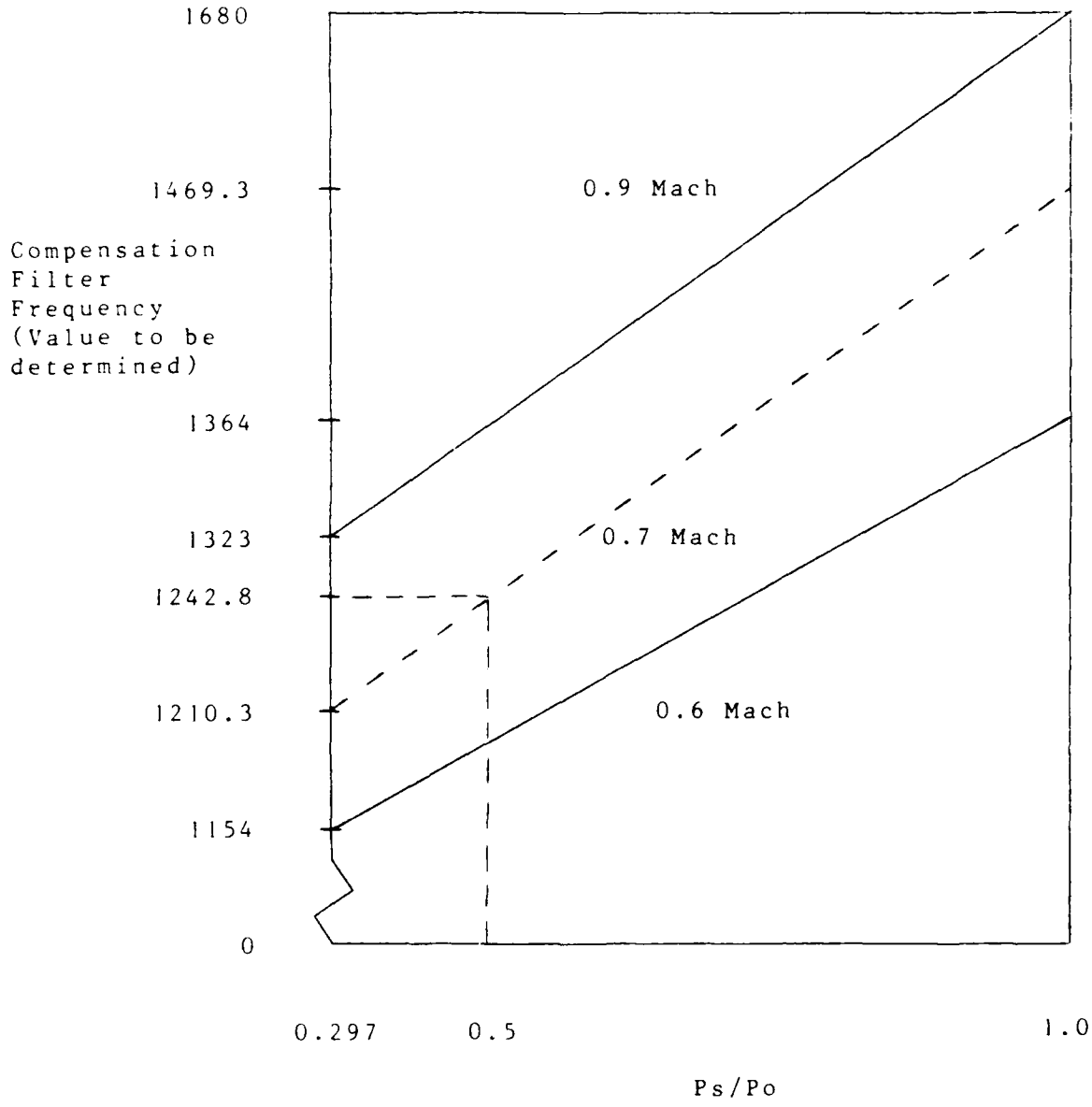


Figure 4. Frequency Compensation Filter Gain Table (Not to scale) (Henley and others, 1984:E-7 to E-8)

Appendix C: Sample Test Results

This appendix presents the results obtained when the developed benchmarks were run on two 1750A processors using two Ada compilers.

Results Evaluation Criteria

Variability in measurements requires that a statistical analysis be performed to insure that there is a true difference between the systems being compared. Memory space used did not vary from one compilation to the next. Execution speed varied only in the two control law benchmarks. To compensate for this variability the small sample t-test with significance level $\alpha = .05$ was used. The following equations are used in inferring differences between two sample means (Mendenhall, 1975:222-227):

$$\text{pooled estimator} \quad s^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2} \qquad (1)$$

$$\text{test statistic} \quad t = \frac{(\bar{y}_1 - \bar{y}_2) - D_0}{s\,(1/n_1 + 1/n_2)^{1/2}} \qquad (2)$$

$$\text{confidence interval} = (\bar{y}_1 - \bar{y}_2) \quad t_{\alpha/2}\, s(1/n_1 + 1/n_2)^{1/2} \qquad (3)$$
$$(\mu_1 - \mu_2)$$

Variables are: n_1, s_1^2, $\bar{y}_1$ - number of samples, variance,

and mean value under condition 1, n_1, s_1, y_1 - number of samples, variance, and mean value under condition 2. The value of n_1 and n_2 is 10 for all cases, chosen based on initial results which showed small variances allowing a small number of samples to be chosen without invalidating results. Because it is desired to determine if there is a difference between various implementations, the null hypothesis is $\mu_1 = \mu_2$ and the alternate hypothesis is $\mu_1 \neq \mu_2$. The null hypothesis will be rejected whenever there is a statistically significant difference between the compared implementations. Finally, the rejection/non-rejection of the null hypothesis and the confidence interval have a 95 percent assurance that they are correct based on $\alpha = .05$ for each individual test.

Control Laws

The first control law test implemented the control laws using global variables for all data. Results are shown in Table IV. The results show that Processor 2 did not run the benchmark (indicated by an asterisk "*"). The processor stopped executing after several hundred instructions but no error was reported and none could be found after a lengthy search. During this research and concurrent research efforts Processor 2 was found to have many errors (Pitarys, 1987). From the results shown in Table IV Compiler 1 is superior to Compiler 2 in both execution speed and code size generated

Table IV. Results of Control Law Test With Global Variables and Constants

	SPEED	
	Compiler 1	Compiler 2
Processor 1	5683.0 usecs	9005.2 usecs
Processor 2	*	*
	CODE SIZE	
	6550 bytes	9855 bytes

* benchmark did not execute

and thus is judged to be better suited for control law computation using global data. With $s_1^2 = 2.0$ and $s_2^2 = 0.17$, $t = 7142.9$ and $t_{\alpha/2} = 2.101$. Because $t > t_{\alpha/2}$ the hypothesis that the means are equal is rejected, and the conclusion is that there is a difference between the compiler/run-time systems. There is 95 percent confidence that the means' difference is 3322.2 ± 0.9 microseconds.

The second test implemented the same control laws using local variables wherever possible. Results are shown in Table V. Processor 2 again did not run the benchmark and reported a numeric error. Execution was halted just prior to the instruction causing the numeric error and values examined to determine if there was a condition which would cause the error. There were no values present which should have caused the error during the next instruction. The next

Table V. Results of Control Law Test With Local Variables and Constants

	SPEED	
	Compiler 1	Compiler 2
Processor 1	6719.8 usecs	9900.7 usecs
Processor 2	*	*

* benchmark did not execute

instruction was executed and a numeric error raised when no error should have been raised. Compiler 1 again was superior to Compiler 2 in speed. Code size is the same as the previous test. With $s_1^2 = 9.95$ and $s_2^2 = 1.12$, $t = 3026.7$ and $t_{\alpha/2} = 2.101$. Because $t > t_{\alpha/2}$ the hypothesis that the means are equal is rejected, and the conclusion is that there is a difference between the compiler/RTS. There is 95 percent confidence that the means' difference is 3180.9 ± 2.2 microseconds.

This test also provided the opportunity to determine whether coding style effected execution speed. For Compiler 1, $s_1^2 = 2.0$ and $s_2^2 = 9.95$, $t = 950.0$ and $t_{\alpha/2} = 2.101$. Because $t > t_{\alpha/2}$ the hypothesis that the means are equal is rejected, and the conclusion is that coding style does affect execution speed. There is 95 percent confidence that the difference is 1036.8 ± 2.3 microseconds. For Compiler 2 $s_1^2 = 0.17$ and $s_2^2 = 1.12$, $t = 2501.0$ and $t_{\alpha/2} = 2.101$.

Because $t > t_{\alpha/2}$ the hypothesis that the means are equal is rejected, and the conclusion is that coding style does affect execution speed. There is 95 percent confidence that the difference is 895.0 ± 0.7 microseconds.

Gain Table Interpolation

The gain table access and interpolation benchmark performs a double interpolation between data points. Results are shown in Table VI. There was no variance in the results and thus no compensation for statistical variation was needed. The results show Compiler 1 to be the superior in both execution speed and generated code. Based on the results from Compiler 1, Processor 1 is superior in speed of execution. This result was expected because Processor 1 is rated at roughly twice the speed of Processor 2.

Table VI. Gain Table Interpolation

	SPEED	
	Compiler 1	Compiler 2
Processor 1	160 usecs	213 usecs
Processor 2	310 usecs	*
	CODE SIZE	
	140 bytes	186 bytes

* benchmark did not execute

Redundancy Management

The redundancy management benchmark implements the leading-edge flap (LEF) monitor portion of the AFTI/F-16 redundancy management function. This test was run in two versions to provide a more robust evaluation: the first started with no monitored systems failed and none detected during execution; the second had some initial system failures and detected additional failures during execution. Tables VII and VIII show the results. There was no variance during the test runs so no statistical compensation was required. For both versions of the test Compiler 1 was superior to Compiler 2 in both generated code and execution speed. As noted in the previous section Processor 1 is faster than Processor 2, and the results bear this out.

Table VII. LEF Monitor With No Failures Detected During Execution

	SPEED	
	Compiler 1	Compiler 2
Processor 1	381 usecs	423 usecs
Processor 2	726 usecs	*
	CODE SIZE	
	713 bytes	765 bytes

* benchmark did not execute

Table VIII. LEF Monitor With Failures Detected During Execution

	SPEED	
	Compiler 1	Compiler 2
Processor 1	508 usecs	544 usecs
Processor 2	990 usecs	*

* benchmark did not execute

Bibliography

Ada Information Clearinghouse. Ada Newsletter, 5: April, 1987.

Barfield, Finley. Flight Control System Engineer. Personal Interviews. AFWAL/FIGI, Wright-Patterson AFB OH, March - October 1987.

Bassman, Mitchell J. and others. "An Approach for Evaluating the Performance Efficiency of Ada Compilers," Ada Letters, 5: September - October 1985.

Bennell, Nicholas and others. Benchmarking: Computer Evaluation and Measurement. Washington, D.C.: Hemisphere Publishing Corporation, 1975.

Booch, Grady. Software Engineering With Ada. Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc., 1983.

Clapp, Russell and others. "Toward Real-Time Performance Benchmarks for Ada," Communications of the ACM, 29:760-778 (August, 1986).

Craine, Capt David. Ada Compiler Evaluation Techniques for Real-time Avionics Applications. MS Thesis, AFIT/GCS/MA/86D-3. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1986.

Department of Defense. Computer Programming Language Policy. DOD Directive 3405.1. Washington: Government Printing Office, 2 April 1987.

Department of Defense. Military Standard: Ada Programming Language. ANSI/MIL-STD-1815A. Washington: Department of Defense, 22 January 1983.

Department of Defense. Military Standard: Sixteen-Bit Computer Instruction Set Architecture. MIL-STD-1750A. Washington: U.S. Government Printing Office, 21 May 1982.

DeThomas, Tony. Flight Control Sysstems Engineer. Personal Interviews. AFWAL/FIGX, Wright-Patterson AFB OH, July - October 1987.

Henley, G. D. and others. AFTI/F-16 Development and Integration Program; DFCS Phase Final Technical Report, Volume 3, Part 3, Contract F33615-78-C-3022. Fort Worth, Texas: General Dynamics, December 1984.

Hicks, Brian. Computer Engineer, F-16 SPO. Personal Interviews. ASD/YPES, Wright-Patterson AFB OH, October 1987.

Hook, Audrey and others. User's Manual for the Prototype Ada Compiler Evaluation Capability (ACEC). Alexandria, Virginia: Institute for Defense Analyses, October 1985.

Johnston, Ann. Flight Controls Engineer. Telephone Interviews. General Dynamics, Fort Worth, Texas, August - October 1987.

Joslin, Lt Paul. Flight Control Software Manager. Personal Interviews. AFWAL/FIGI, Wright-Patterson AFB OH, March - August 1987.

Joyce, Capt Daniel O. Validating and Evaluating Ada's Representation Clauses on MIL-STD-1750A Architecture. MS Thesis, AFIT/GCS/MA/87D-6. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1987.

Lahn, T. G. and others. Some Views on the Use of Ada for Digital Flight Control Systems. Unpublished report. Honeywell Military Avionics Division, Minneapolis, Minnesota, undated.

Landy, R. J. and others. Ada Based Integrated Control System II (Draft). St. Louis, Missouri: McDonnell Douglas Corp., December 1986.

Lewantowicz, Lt Col Zdzislaw. Instructor and Deputy Head Department of Electrical and Computer Engineering. Personal Interview. AFIT/ENG, Wright-Patterson AFB OH, October 1987.

Mellby, John. Control Systems Engineer. Telephone Interview. Texas Instruments, Houston, Texas, 16 July 1987.

Mendenhall, William. Introduction to Probability and Statistics (Fourth Edition). North Scltuate, Massachusetts: Duxbury Press, 1975.

Pitarys, Lt Marc. Avionics System Engineer. Personal Interviews. AFWAL/AAAT, Wright-Patterson AFB OH, March - August 1987.

Ramage, James K. and others. " AFTI/F-16 Digital Flight Control System Development Status," Presented at the Fourth AIAA/ IEEE Digital Avionics System Conference. November 17-19, 1981.

Startzman, Ty. Control Systems Engineer. Telephone Interviews. Boeing Aerospace, Wichita, Kansas, July - August 1987.

Squire, Jon. Chairman, PIWG of the ACM. Telephone Interview. Westinghouse Defense and Electronics Center, Baltimore, Maryland, 29 June 1987.

-----. PIWG Benchmark Programs contained in the directory <INFO-ADA.PIWG> on ADA20.ISI.EDU on the ARPANET. Baltimore, Maryland: Westinghouse Defense and Electronics Center, August 1986.

Toolean, Bill. Flight Systems Engineer. Telephone Interviews. Grumman Aircraft, Long Island, New York, July - September 1987.

University of Colorado at Colorado Springs. IFCC Software: Ada Language, Contract F33615-84-K-3602. Colorado Springs, Colorado: University of Colorado at Colorado Springs, January 1986.

Weicker, Reinhold P. "Dhrystone: A Synthetic Systems Programming Benchmark," Communications of the ACM, 27: 1013-1027 (October, 1984).

Westermeier, T. F. and H. E. Hansen. "The Use of Ada in Digital Flight Control Systems," AIAA Journal, 22: 597-603 (August, 1985).

Witt, Capt Donald J. Using Ada in the Real-Time Avionics Environment: Issues and Answers. MS Thesis, AFIT/GCS/ MA/85D-6. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1985.

Yousey, W. J. and others. AFTI/F-16 Development and Integration Program; DFCS Phase Final Technical Report, Volume 3, Part 2, Contract F33615-78-C-3022. Fort Worth, Texas: General Dynamics, December 1984.

VITA

John D. Klemens was born on 15 September 1956 in Louisville, Ohio. He graduated from Louisville High School in 1974. He then attended Ohio University from 1974 to 1978 where he received a Bachelor of Science Degree in Computer Science. While at Ohio University, he enrolled in Army ROTC and was commissioned a second lieutenant of Field Artillery on 10 June 1978. After attending basic artillery school he served in Germany for three years in various artillery positions. He returned to the United States, attended advanced artillery school, served in various positons concluding with a Battery command at Fort Polk, Louisiana until July 1986. He then reported to the Air Force Institute of Technology at Wright-Patterson AFB, Ohio.

Permanent address: 1118 East Broad Street
Louisville, Ohio 44641

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENC/87D-3	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO
11. TITLE (Include Security Classification) See Box 19	
12. PERSONAL AUTHOR(S) ohn D. Klemens, B.S., CPT, USA	
13a. TYPE OF REPORT MS Thesis 13b. TIME COVERED FROM _____ TO _____ 14. DATE OF REPORT (Year, Month, Day) 1987 December 15. PAGE COUNT 74	
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
12	08	
01	04	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Language-Programming Languages-Ada
Flight Control Systems-Control Systems

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

Title: Examination of the Effects of Using Ada in Flight Control Software

Thesis Chairman: Richard R. Gross, Lt Col, USAF
Assistant Dean, School of Engineering

Lynn Wilson 31 Dec 87

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Richard R. Gross, Lt Col, USAF	22b. TELEPHONE (Include Area Code) (513) 255-4372 22c. OFFICE SYMBOL AFIT/ENA

DD Form 1473, JUN 86 *Previous editions are obsolete*

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Abstract

This research characterizes flight control software in terms of the Ada (TM) constructs used and of the performance characteristics that determine the suitability of a particular compiler/processor system for flight control software. A new set of three flight control software benchmarks based on this characterization was evaluated on two MIL-STD-1750A processors using two Ada compilers. Results show that compiler/processor combinations can be compared for their ability to implement flight control software in execution speed and memory size required. The benchmarks provide time and space requirements for a typical sample of flight control software.

END

DATE

3-88

DTIC