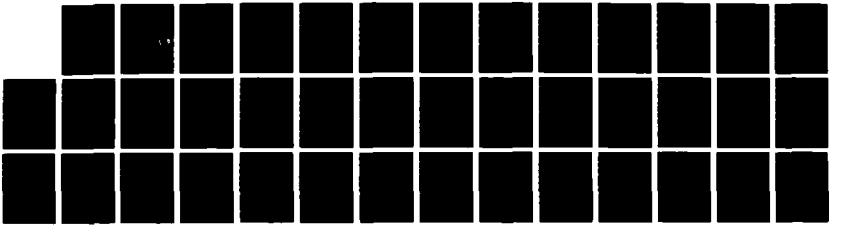
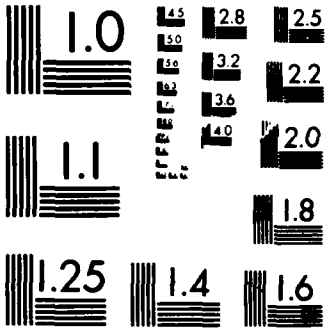
AD-A188 070 ADA (TRADENAME) COMPILER VALIDATION SUMMARY REPORT 1/1
HARRIS CORPORATION HAR..(U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI.. 20 JUN 86
UNCLASSIFIED F/G 12/5 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

(2)

AD-A180 078

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETEING FORM
1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: Harris Corporation, HARRIS Ada Compiler, Version 1.0, Harris H700 and H60	5. TYPE OF REPORT & PERIOD COVERED 28 JUN 1986 to 28 JUN 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Ada Validation Facility ASD/SIOL Wright-Patterson AFB, OH 45433-6503	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 28 JUN 1986 13. NUMBER OF PAGES 36
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) Wright-Patterson	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

UNCLASSIFIED

DTIC ELECTE MAY 07 1987 S D E

18. SUPPLEMENTARY NOTES

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

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

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

See Attached.

DD FORM 1473 1 JAN 73 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ada® Compiler Validation Summary Report:

Compiler Name: HARRIS Ada Compiler, Version 1.0

Host Computers:	Target Computers:
Harris H700 under VOS 5.1	Harris H700 under VOS 5.1
and	and
Harris H60 under VOS 5.1	Harris H60 under VOS 5.1

Testing Completed 28 JUN 1986 Using ACVC 1.7

This report has been reviewed and is approved.

Georgeanne C Chitwood

Ada Validation Facility
Georgeanne Chitwood
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

for JFK

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

AVF Control Number: AVF-VSR-43.1086

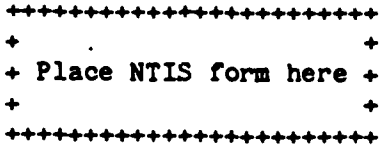
Ada® COMPILER
VALIDATION SUMMARY REPORT:
Harris Corporation
HARRIS Ada Compiler, Version 1.0
Harris H700 and H60

Completion of On-Site Validation:
28 JUN 1986

Prepared By:
Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).


Place NTIS form here

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the HARRIS Ada Compiler, Version 1.0, using Version 1.7 of the Ada® Compiler Validation Capability (ACVC).

The validation process includes submitting a suite of standardized tests (the ACVC) as inputs to an Ada compiler and evaluating the results. The purpose is to ensure conformance of the compiler to ANSI/MIL-STD-1815A Ada by testing that it properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by ANSI/MIL-STD-1815A. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, or during execution.

On-site testing was performed 24 JUN 1986 through 28 JUN 1986 at Harris Corporation, Ft. Lauderdale FL, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The HARRIS Ada Compiler, Version 1.0, is hosted on a Harris H700 operating under VOS 5.1. It is also hosted on a Harris H60 operating under VOS 5.1.

The results of validation are summarized in the following table:

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	815	1051	17	9	21	1981
Failed	0	0	0	0	0	0	0
Inapplicable	0	9	269	0	2	2	282
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

©Ada is a registered trademark of the United States Government (Ada Joint Program Office).

There were 16 withdrawn tests in ACVC Version 1.7 at the time of this validation attempt. A list of these tests appears in Appendix D.

Some tests demonstrate that some language features are or are not supported by an implementation. For this implementation, the tests determined the following:

- SHORT_INTEGER, LONG_INTEGER, SHORT_FLOAT, and LONG_FLOAT are not supported.
- Representation specifications for noncontiguous enumeration representations are supported.
- Generic unit specifications and bodies can be compiled in separate compilations.
- Pragma INLINE is not supported for procedures or functions.
- The package SYSTEM is used by package TEXT_IO.
- Modes IN_FILE and OUT_FILE are supported for sequential I/O.
- Instantiation of the package SEQUENTIAL_IO with unconstrained array types is supported.
- Instantiation of the package SEQUENTIAL_IO with unconstrained record types with discriminants is supported.
- RESET and DELETE are supported for sequential and direct I/O.
- Modes IN_FILE, INOUT_FILE, and OUT_FILE are supported for direct I/O.
- Instantiation of package DIRECT_IO with unconstrained array types and unconstrained types with discriminants is supported.
- Dynamic creation and deletion of files are supported.
- Only one internal file can be associated with the same external file.
- Illegal file names can exist.

ACVC Version 1.7 was taken on-site via three magnetic tapes to Harris Corporation, Ft. Lauderdale FL. All tests, except the withdrawn tests and any executable tests that make use of a floating-point precision greater than SYSTEM.MAX_DIGITS, were compiled on a Harris H700. Class A, C, D, and E tests were executed on a Harris H700. A subset of the tests was compiled, linked, and executed on the Harris H60 computer.

On completion of testing, execution results for Class A, C, D, or E tests were examined. Compilation results for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. The test results produced from the subset testing were identical to the results generated by the compiler operating on the Harris H700.

The AVF identified 2021 of the 2279 tests in Version 1.7 of the ACVC as potentially applicable to the validation of the HARRIS Ada Compiler, Version 1.0. Excluded were 242 tests requiring a floating-point precision greater than that supported by the implementation and the 16 withdrawn tests. After the 2021 tests were processed, 40 tests were determined to be inapplicable. The remaining 1981 tests were passed by the compiler.

The AVF concludes that these results demonstrate acceptable conformance to ANSI/MIL-STD-1815A.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard (ANSI/MIL-STD-1815A). Any implementation-dependent features must conform to the requirements of the Ada Standard. The entire Ada Standard must be implemented, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to ANSI/MIL-STD-1815A, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from limitations imposed on a compiler by the operating system and by the hardware. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

VSRs are written according to a standardized format. The reports for several different compilers may, therefore, be easily compared. The information in this report is derived from the test results produced during validation testing. Additional testing information as well as details which are unique for this compiler are given in section 3.7. The format of a validation report limits variance between reports, enhances readability of the report, and minimizes the delay between the completion of validation testing and the publication of the report.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any unsupported language constructs required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to policies and procedures established by the Ada Validation Organization (AVO). Testing was conducted from 24 JUN 1986 through 28 JUN 1986 at Harris Corporation, Ft. Lauderdale FL.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformances to ANSI/MIL-STD-1815A other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139
1211 S. Fern, C-107
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard
Alexandria VA 22311

1.3 RELATED DOCUMENTS

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformance of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting policies and procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformance to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
LMC	The Language Maintenance Committee whose function is to resolve issues concerning the Ada language.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.

Test	A program that evaluates the conformance of a compiler to a language specification. In the context of this report, the term is used to designate a single ACVC test. The text of a program may be the text of one or more compilations.
Withdrawn test	A test found to be inaccurate in checking conformance to the Ada language specification. A withdrawn test has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformance to ANSI/MIL-STD-1815A is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Special program units are used to report the results of the Class A, C, D, and E tests during execution. Class B tests are expected to produce compilation errors, and Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. (However, no checks are performed during execution to see if the test objective has been met.) For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a message indicating that it has passed.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntactical or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT-APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters (e.g., the number of identifiers permitted in a compilation, the number of units in a library, and the number of nested loops in a subprogram body), a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT-APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report results. It also provides a set of identity functions used to defeat some compiler optimization strategies and force computations to be made by the target computer instead of by the compiler on the host computer. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard.

The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

Some of the conventions followed in the ACVC are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values. The values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformance to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The nonconformant tests are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configurations:

Compiler: HARRIS Ada Compiler, Version 1.0

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computers:

Machine:	Harris H700	Harris H60
Operating System:	VOS 5.1	VOS 5.1
Memory Size:	6 megabytes	6 megabytes

Target Computers:

Machine:	Harris H700	Harris H60
Operating System:	VOS 5.1	VOS 5.1
Memory Size:	6 megabytes	6 megabytes

2.2 CERTIFICATE INFORMATION

Configurations:

Compiler: HARRIS Ada Compiler, Version 1.0

Test Suite: Ada Compiler Validation Capability, Version 1.7

Certificate Date: 3 September 1986

Host Computers:

Machine:	Harris H700	Harris H60
Operating System:	VOS 5.1	VOS 5.1

Target Computers:

Machine:	Harris H700	Harris H60
Operating System:	VOS 5.1	VOS 5.1

2.3 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Nongraphic characters.

  Nongraphic characters are defined in the ASCII character set but are not permitted in Ada programs, even within character strings. The compiler correctly recognizes these characters as illegal in Ada compilations. The characters are printed in the output listing. (See test B26005A.)

- Capacities.

  The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A through D55A03H, D56001B, D64005E through D64005G, and D29002K.)

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

. Predefined types.

This implementation does not support any additional predefined types in the package STANDARD. (See tests B86001CR, B86001CS, B86001CP, B86001CQ, and B86001DT.)

. Based literals.

An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

. Array types.

When an array type is declared with an index range exceeding the INTEGER'LAST values and with a component that is a null BOOLEAN array, this compiler raises NUMERIC_ERROR when the type is declared. (See tests E36202A and E36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR either when declared or assigned. Alternately, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype.

In assigning two-dimensional array types, the entire expression does not appear to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

  During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

  In assigning record types with discriminants, the entire expression appears to be evaluated before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

  In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

  In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

  All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

- Functions.

  The declaration of a parameterless function with the same profile as an enumeration literal in the same immediate scope is rejected by the implementation. (See test E66001D.)

- Representation clauses.

  Enumeration representation clauses are supported. (See test BC1002A.)

- Pragmas.

  The pragma INLINE is not supported for procedures nor is it supported for functions. (See tests CA3004E and CA3004F.)

- Input/output.

  The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests CE2201D, CE2201E, and CE2401D.)

Only one internal file can be associated with each external file for sequential, direct, and text I/O. (See tests CE2107A .. D (4 tests), CE2107F, and CE3111A .. E (5 tests).)

An existing text file can be opened in OUT_FILE mode, but it cannot be created in OUT_FILE mode nor in IN_FILE mode. (See test EE3102C.)

Temporary sequential and direct files are given a name. Temporary files given names are deleted when they are closed. (See tests CE2108A and CE2108C.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

The AVF identified 2021 of the 2279 tests in Version 1.7 of the ACVC as potentially applicable to the validation of the HARRIS Ada Compiler, Version 1.0. Excluded were 242 tests requiring a floating-point precision greater than that supported by the implementation and the 16 withdrawn tests. After they were processed, 40 tests were determined to be inapplicable. The remaining 1981 tests were passed by the compiler.

The AVF concludes that the testing results demonstrate acceptable conformance to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	68	815	1051	17	9	21	1981
Failed	0	0	0	0	0	0	0
Inapplicable	0	9	269	0	2	2	282
Withdrawn	0	4	12	0	0	0	16
TOTAL	68	828	1332	17	11	23	2279

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	96	201	272	241	160	97	155	198	99	28	216	218	1981
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	20	106	122	6	1	0	6	1	6	0	0	14	282
Withdrawn	0	1	4	0	0	0	1	2	6	0	1	1	16
TOTAL	116	308	398	247	161	97	162	201	111	28	217	233	2279

3.4 WITHDRAWN TESTS

The following tests have been withdrawn from the ACVC Version 1.7:

B4A010C	C41404A	CA1003B
B83A06B	C48008A	CA3005A through CA3005D (4 tests)
BA2001E	C4A014A	CE2107E
BC3204C	C92005A	
C35904A	C940ACA	

See Appendix D for the test descriptions.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 282 tests were inapplicable for the reasons indicated:

- C34001D, B52004E, B55B09D, B86001CR, and C55B07B use SHORT_INTEGER which is not supported by this compiler.
- C34001E, B52004D, B55B09C, B86001CS, and C55B07A use LONG_INTEGER which is not supported by this compiler.
- C34001F, C35702A, and B86001CP use SHORT_FLOAT which is not supported by this compiler.
- C34001G, C35702B, and B86001CQ use LONG_FLOAT which is not supported by this compiler.

- C64103A requires certain predefined operations to raise NUMERIC_ERROR. However, the Harris Ada compiler follows AI-387's recommendation that CONSTRAINT_ERROR be raised instead. Analysis of the output confirmed that CONSTRAINT_ERROR was raised at all appropriate places.

- B86001DT requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

- C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATIONS's base type. This is not the case for this implementation.

- CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

- CE2107A, CE2107B, CE2107D, CE2107F, CE2110B, CE2111D, CE2111H, CE3111A through CE3111E, CE3114B, and CE3115A are inapplicable because multiple internal files cannot be associated with the same external file.

- 242 tests were not processed because SYSTEM.MAX_DIGITS was nine. These tests were:

C24113F through C24113Y (20 tests)
C35705F through C35705Y (20 tests)
C35706F through C35706Y (20 tests)
C35707F through C35707Y (20 tests)
C35708F through C35708Y (20 tests)
C35802F through C35802Y (20 tests)
C45241F through C45241Y (20 tests)
C45321F through C45321Y (20 tests)
C45421F through C45421Y (20 tests)
C45424F through C45424Y (20 tests)
C45521F through C45521Z (21 tests)
C45621F through C45621Z (21 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then

compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for 19 Class B tests.

B24104A	B24104B	B24104C
B2A003A	B2A003B	B2A003C
B33004A	B37201A	B38008A
B41202A	B44001A	B64001A
B67001A	B67001B	B67001C
B67001D	B910ABA	B95001A
B97101E		

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.7 produced by the HARRIS Ada Compiler, Version 1.0, on the H700 was submitted to the AVF by the applicant for prevalidation review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests. No prevalidation materials were received for the Harris H60.

3.7.2 Test Method

Testing of the HARRIS Ada Compiler using ACVC Version 1.7 was conducted on-site by a validation team. The configurations consisted of a Harris H700 host and target computer operating under VOS 5.1 and a Harris H60 host and target computer operating under VOS 5.1.

Three magnetic tapes containing ACVC Version 1.7 were taken on-site by the validation team. The magnetic tapes contained all tests applicable to this validation, as well as all tests inapplicable to this validation, except for any Class C tests that require floating-point precision exceeding the maximum value supported by the implementation. Tests that make use of values that are specific to an implementation were customized before being written to the magnetic tapes.

The contents of the magnetic tapes were loaded directly onto two Harris H700 computers. After the test files were loaded to disk, the files were grouped by test class and moved into separate directories. Support units, REPORT and CHECK_FILE, were compiled on each of the H700 computers and were determined to be operating correctly. The full set of tests was compiled and run in batch mode in two job queues on each H700 during validation

testing. Class B and C tests were run in smaller groups according to chapter within class. Larger chapters were further divided into three and four smaller groups of tests. Test results were printed and reviewed by the validation team.

A subset of the ACVC 1.7 was run on the Harris H60, which is architecturally identical to the Harris H700. The subset of 60 tests consisted of five tests selected at random from the classes of tests within each chapter. The 60 tests were taken on-site via a magnetic tape and copied to cartridge tape. The contents of the cartridge tape were loaded to the H60, and the tests were compiled, linked, and executed (as appropriate). The test results were printed, and the validation team found them identical to the results produced by the H700.

The compiler was tested using command scripts provided by Harris Corporation. These scripts were reviewed by the validation team. The following switches were in effect for testing:

All tests were compiled with the options:

```
-v    (verbose)
-el   (error listing)
```

Those which compile and link used:

```
-M <unit_name>  (main unit name)
-o <a.out>      (output_filename is "a.out")
```

For the main library units that execute only:

```
-o <a.out>    (output_filename is "a.out")
```

Under the operating system VOS 5.1, two settings were used for the testbed program size. All but five tests used 512 words of memory. Five tests required 2047 words of memory:

```
C41203B
C45526A
C52102B
C52102D
CE3604A
```

Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

The validation team arrived at Harris Corporation, Ft. Lauderdale FL, on 24 JUN 1986 and departed after testing was completed on 28 JUN 1986.

APPENDIX A

COMPLIANCE STATEMENT

Harris Corporation has submitted the following compliance statement concerning the HARRIS Ada Compiler.

Compliance Statement

Base Configuration:

Compiler: HARRIS Ada Compiler, Version 1.0

Test Suite: Ada Compiler Validation Capability, Version 1.7

Host Computer:

Machine(s):	Harris H700	Harris H60
Operating System:	VOS 5.1	VOS 5.1

Target Computer:

Machine(s):	Harris H700	Harris H60
Operating System:	VOS 5.1	VOS 5.1

Harris Corporation has made no deliberate extensions to the Ada language standard.

Harris Corporation agrees to the public disclosure of this report.

Harris Corporation agrees to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

Wendell Norton Date: 6-26-86

Harris Corporation
Wendell Norton
Director of Contracts

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the HARRIS Ada Compiler, Version 1.0, are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Two other sections, package STANDARD and file naming conventions, are also included in this appendix.

1 Pragmas

1.1 Implementation-Dependent Pragmas

PRAGMA CONTROLLED is recognized by the implementation but has no effect in this release.

PRAGMA INLINE is recognized by the implementation but has no effect in this release.

PRAGMA INTERFACE is recognized by the implementation and supports calls to C and FORTRAN language functions with an optional link name for the subprogram. The Ada specifications can be either functions or procedures. All parameters must have mode IN.

For C, the types of parameters and the result type for functions must be scalar, access, or the predefined type ADDRESS defined in the package SYSTEM. Record and array objects may be passed by reference using the ADDRESS attribute.

For FORTRAN, all parameters are passed by reference; the parameter types must have the type ADDRESS defined in the package SYSTEM. The result type for a FORTRAN function must be a scalar type. Care should be taken when using tasking and FORTRAN functions. Since FORTRAN is not reentrant we suggest that an Ada controller task should be used to access FORTRAN functions.

The optional link name enables calling a function whose name is defined in another language, allowing characters in the name that are not allowed in an Ada identifier. Case sensitivity can then be preserved. Without the optional link name, the Ada compiler converts all C interface names to lower case and all FORTRAN interface names to upper case. For instance, the following example generates a reference for $Var1 with no case or other changes: pragma INTERFACE (language_name, Var1, "$Var1") ;

PRAGMA MEMORY_SIZE is recognized by the implementation, but has no effect. The implementation does not allow the package SYSTEM to be modified by means of pragmas; however, the same effect can be achieved by recompiling SYSTEM with altered values.

PRAGMA OPTIMIZE is recognized by the implementation but has no effect in this release.

PRAGMA PACK will cause the compiler to choose a non-aligned representation for composite types. In the current release, it will not cause objects to be packed at the bit level.

PRAGMA SHARED is recognized by the implementation but has no effect in this release.

PRAGMA STORAGE_UNIT is recognized by the implementation but has no effect. The implementation does not allow the package SYSTEM to be modified by means of pragmas; however, the same effect can be achieved by recompiling SYSTEM with altered values.

PRAGMA SUPPRESS is recognized by the implemention and applies from the point of occurrence to the end of the innermost enclosing block. The double parameter form of the pragma, with a name of an object, type, or subtype is recognized, but has no effect.

PRAGMA SYSTEM_NAME is recognized by the implementation but has no effect. The implementation does not allow the package SYSTEM to be modified by means of pragmas; however, the same effect can be achieved by recompiling SYSTEM with altered values.

1.2 Implementation-Defined Pragmas

PRAGMA SHARE_BODY is used to indicate a desire to share or not share an instantiation. The pragma may reference the generic unit or the instantiated unit. When it references a generic unit, it sets sharing on/off for all instantiations of that generic, unless overridden by specific SHARE_BODY pragmas for individual instantiations. When it references an instantiated unit, sharing is on/off only for that unit. The default is to share all generics that can be shared, unless the unit uses PRAGMA IN_LINE.

PRAGMA SHARE_BODY is only allowed in the following places: immediately within a declarative part, immediately within a package specification, or after a library unit in a compilation, but before any subsequent compilation unit.

The form of this pragma is:
pragma SHARE_BODY (generic_name, boolean_literal)
Note that a parent instantiation is independent of any individual instantiation, therefore recompilation of a generic with different parameters has no effect on other compilations that reference it. The unit that caused compilation of a parent instantiation need not be referenced in any way by subsequent units that share the parent instantiation.

Sharing generics causes a slight execution time penalty because all type attributes must be indirectly referenced (as if an extra calling argument were added). However, it substantially reduces compilation time in most circumstances and reduces program size.

2 Implementation-Dependent Attributes

There are no implementation-dependent attributes in HAPSE.

3 Specification of the package SYSTEM

```
package SYSTEM is

   type ADDRESS is private  ;
   type NAME is ( harris_vue ) ;

   SYSTEM_NAME                    : constant NAME := harris_vue ;
```

```
   -- System-Dependent Constraints

   STORAGE_UNIT                    : constant := 8 ;
   MEMORY_SIZE                     : constant := 6_291_456 ;

   -- System-Dependent Named Numbers

   MIN_INT                         : constant := - 8_388_608 ;
   MAX_INT                         : constant := 8_388_607 ;
   MAX_DIGITS                      : constant := 9 ;
   MAX_MANTISSA                    : constant := 38 ;
   FINE_DELTA                      : constant := 2.0**(-30) ;
   TICK                            : constant := 0.01 ;

   -- Other System-dependent Declarations

   subtype PRIORITY is INTEGER range 0 .. 23 ;

   MAX_REC_SIZE : integer := 32_767 * 3 ;

private

   type ADDRESS is new INTEGER;

end SYSTEM ;
```

4 Restrictions on Representation Clauses

4.1 Pragma PACK

Bit packing is not supported. In the presence of pragma PACK, components of composite types are packed to the nearest whole STORAGE_UNIT.

4.2 Length Clauses

The specification T'SIZE is not supported. The specification T'SMALL is not supported. The specification T'STORAGE_SIZE is supported.

4.3 Record Representation Clauses

Component clauses must specify alignment on multiples of 3 STORAGE_UNIT boundaries.

4.4 Address Clauses

Address clauses and interrupts are not supported.

5 Other Representation Implementation-Dependencies

Change of representation is not supported for record types.

The ADDRESS attribute is not supported for the following entities: static constants; packages; tasks; labels; and entries.

Machine code insertions are not supported.

6 Conventions for Implementation-Generated Names

There are no implementation generated names.

7 Interpretation of Expressions in Address Clauses

Address clauses and interrupts are not supported.

8 Restrictions on Unchecked Conversions

The predefined generic function UNCHECKED_CONVERSION cannot be instantiated with a target type that is an unconstrained array type or an unconstrained record type with discriminants.

9 Implementation Characteristics of I/O Packages

9.1 Interpretation of Strings as Applied to External Files

Strings that contain names of external files are interpreted in the following manners for each of the respective external file environments.

VUE external files: file names may be composed of up to 512 characters of the ASCII character set except for "/", ascii.nul, and non-printable characters. Further, the first character of a file must be alpha-numeric, "." or "_". If the "/" character is encountered in a string, it is interpreted as a separater between file names that specify VUE directories.

VOS external files: file names are composed of a 1 to 8 character qualifier plus a 1 to 8 character areaname. The first character of the areaname must be alphabetic. The remaining characters comprising the areaname may be drawn from the following set of characters: A-Z, 0-9, :, #, -, /, . and " ". The qualifier portion of a file name is optional. If specified, it must be comprised of an account portion, a name portion, and an asterisk. The account portion may be null, or 1-4 characters from the following set: 0-9. The name portion may be null, or 1-4 characters from the following set: A-Z, 0-9. The name portion may not be null if the account portion is not null. If lower case letters are encountered in the string they are converted to upper case.

9.2 Interpretation of Strings as Applied to Form Parameters

The OPEN and CREATE I/O procedures accept FORM parameters, in order to specify implementation dependent attributes of files. The HAPSE implementation supports the attributes described below. These attributes may be specified in any order. Blanks may be inserted between attributes, however none are required. No attribute can be specified more than once. All attributes must be specified in uppercase. These attributes are only applicable to CREATE calls. A form string passed to OPEN is ignored.

File Type Attributes

BL	Blocked file
UB	Unblocked file
RA	Random file

These attributes specify the VOS file type of a file to be created. UB is the default for all files. In general, the defaults should not be overrideen for direct and sequential I/O.

Double Buffered Blocking

DB	Defines a BL type file as permanently double buffered

This attribute can only be specified if the file type is BL.

Directory Type

CD	The VOS directory entry for this file is to be kept resident
DD	The VOS directory entry is kept on disc

Access Parameters

PR	PUBLIC READ
PW	PUBLIC WRITE
PD	PUBLIC DELETE
AR	ACCOUNT READ
AW	ACCOUNT WRITE
AD	ACCOUNT DELETE
OW	OWNER WRITE
OD	OWNER DELETE

These attributes determine the access permissions associated with a file. The default access is OW OD. Note that if any access attributes are specified, then only the specified accesses will be granted (i.e. OW OD is not assumed).

File Definition Attributes

A=n	Access level, n = 0-15, VOS access required to access file
B=n	Blocking factor, where n is 1-7 sectors
C=n	Current size, where n is the number of sectors requested
E=n	Eliminate date, where n is the number of days before purging
G=n	Granule size, where n is the number of sectors per granule
M=n	Maximum size, n = number of sectors to which file may expand
P=n	Pack number, n = pack number of pack on which to create file
T=n	Type number, n = 0-7, provided for user file classification

No spaces are allowed between the attribute letter, the equal sign, and the integer value.

9.3 Implementation-Dependent Characteristics of DIRECT_IO

Instantiations of DIRECT_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_REC_SIZE is defined in SYSTEM and can be changed by a program before instantiating DIRECT_IO to provide an upper limit on the record size. In any case, the maximum size supported is 32_768 * 3 * STORAGE_UNIT bits. DIRECT_IO will raise USE_ERROR if MAX_RECORD_SIZE exceeds this absolute limit.

9.4 Implementation-Dependent Characteristics of SEQUENTIAL_IO

Instantiations of SEQUENTIAL_IO use the value MAX_REC_SIZE as the record size (expressed in STORAGE_UNITs) when the size of ELEMENT_TYPE exceeds that value. For example, for unconstrained arrays such as string where ELEMENT_TYPE'SIZE is very large, MAX_REC_SIZE is used instead. MAX_REC_SIZE is defined in SYSTEM and can be changed by a program before instantiating INTEGER_IO to provide an upper limit on the record size. In any case, the maximum size supported is 32_768 * 3 *STORAGE_UNIT bits. SEQUENTIAL_IO will raise USE_ERROR if MAX_REC_SIZE exceeds this absolute limit.

Package STANDARD contains the following declarations:

```
type INTEGER is range -8_388_608 .. 8_388_607;

type FLOAT is digits 9 range
   -2#0.11111111111111111111111111111111#E127 ..
    2#0.11111111111111111111111111111111#E127 ;

type DURATION is delta 2#1.0#E-8 range -2#1.0#E31 ..
    2#11111111111111111111111111111.11111111# ;
```

DURATION'SMALL = 3.90625E-03 seconds

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value is substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier of size MAX_IN_LEN with varying last character.	(1..498 => 'A', 499 => '1')
$BIG_ID2 Identifier of size MAX_IN_LEN with varying last character.	(1..498 => 'A', 499 => '2')
$BIG_ID3 Identifier of size MAX_IN_LEN with varying middle character.	(1..249 => 'A', 250 => '3', 251..499 => 'A')
$BIG_ID4 Identifier of size MAX_IN_LEN with varying middle character.	(1..249 => 'A', 250 => '4', 251..499 => 'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is MAX_IN_LEN characters long.	(1..496 => '0', 497..499 => "298")

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be MAX_IN_LEN characters long.	(1..493 => '0', 494..499 => "69.0E1")
$BLANKS Blanks of length MAX_IN_LEN - 20	(1..479 => ' ')
$COUNT_LAST Value of COUNT'LAST in TEXT_IO package.	8_388_607
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{}~"
$FIELD_LAST Value of FIELD'LAST in TEXT_IO package.	8_388_607
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long.	"./^BAD-CHARACTER"
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	"./CE2102" & (1..254 => 'C')
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST.	10_000_000_000.0
$ILLEGAL_EXTERNAL_FILE_NAME1 Illegal external file name.	"./^ILLEGAL_EXTERNAL_FILE_NAME1"
$ILLEGAL_EXTERNAL_FILE_NAME2 Illegal external file names.	"/no/such/directory/ILLEGAL_EXT_FILE_NAME2"

Name and Meaning	Value
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-8_388_608
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	8_388_607
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST.	-10_000_000_000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	9
$MAX_IN_LEN Maximum input line length permitted by the implementation.	499 (plus line feed character)
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	LONG_LONG_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFD#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. When testing was performed, the following 16 tests had been withdrawn at the time of validation testing for the reasons indicated:

- B4A010C: The object_declaration in line 18 follows a subprogram body of the same declarative part.

- B83A06B: The Ada Standard 8.3(17) and AI-00330 permit the label LAB_ENUMERAL of line 80 to be considered a homograph of the enumeration literal in line 25.

- BA2001E: The Ada Standard 10.2(5) states: "Simple names of all subunits that have the same ancestor library unit must be distinct identifiers." This test checks for the above condition when stubs are declared. However, the Ada Standard does not preclude the check being made when the subunit is compiled.

- BC3204C: The file BC3204C4 should contain the body for BC3204C0 as indicated in line 25 of BC3204C3M.

- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR (instead of CONSTRAINT_ERROR).

- C41404A: The values of 'LAST and 'LENGTH are incorrect in IF statements from line 74 to the end of the test.

- C48008A: This test requires that the evaluation of default initial values not occur when an exception is raised by an allocator. However, the Language Maintenance Committee (LMC) has ruled that such a requirement is incorrect (AI-00397/01).

- C4A014A: The number declarations in lines 19-22 are incorrect because conversions are not static.

- C92005A: At line 40, "/=" for type PACK.BIG_INT is not visible without a USE clause for package PACK.

- C940ACA: This test assumes that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program; however, such an execution order is not required by the Ada Standard, so the test is erroneous.

- CA1003B: This test requires all of the legal compilation units of a file containing some illegal units to be compiled and executed. According to AI-00255, such a file may be rejected as a whole.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- CE2107E: This test has a variable, TEMP_HAS_NAME, that needs to be given an initial value of TRUE.

END

5-87

DTIC