AVF Control Number: EDS19980518OCS02-2.1
DATE COMPLETED
BEFORE ON-SITE: 1 June 1998
AFTER ON-SITE: 14 July 1998

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 980605E2.1-022
OC Systems, Inc.
PowerAda 4.0
IBM RS/6000 Model 7025-F50 under AIX 4.2 =>
Motorola Powerstack MT 603-66 under LynxOS 2.4.0

Prepared By:
Ada Validation Facility
Electronic Data Systems
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton, OH 45424-0593
U.S.A.

TABLE OF CONTENTS

PREFACE

This report documents the validation testing of an Ada 95 implementation. This testing was conducted according to the Ada Compiler Validation Procedures version 5.0 using the Ada Compiler Validation Capability test suite version 2.1, and completed June 5, 1998.

The successful completion of validation testing is the basis for the Ada certification body's issuance of a validation certificate and for subsequent registration of derived implementations. A copy of the validation certificate 980605E2.1-022 and its attachment which were awarded for this validation are presented on the following two pages. Validation testing does not ensure that an implementation has no nonconformities to the Ada 95 standard other than those, if any, documented in this report. The compiler vendor declares that the tested implementation contains no deliberate deviation from the Ada 95 standard; a copy of this Declaration of Conformance is presented immediately after the certificate pages.

This report has been reviewed and approved by the signatories below. These organizations attest that, to the best of their knowledge, this report is accurate and complete; however, they make no warrant, express or implied, that omissions or errors have not occurred.

Ada Validation Facility
Phil Brashear, AVF Manager
Electronic Data Systems
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton, OH 45424-0593
U.S.A.

Ada Validation Organization
Director, Computer and Software
Engineering Division
Institute for Defense Analyses
Alexandria VA 22311
U.S.A.

Ada Joint Program Office
Director
Center for Information Management
Defense Information Systems Agency
Alexandria VA 22041
U.S.A.

(Insert copy of certificate here)

Results Summary for 980605E2.1-022

Specialized Needs Annexes

Note: Tests allocated to these annexes are processed only when the vendor claims support.

SPECIALIZED NEEDS ANNEXES	Total	With-Drawn	Passed	Inappli-cable	Unsup-ported
C Systems Programming & required Section 13 (representation support)	24 161 --- 185	2 1 --- 3	22 160 --- 182	0 0 --- 0	0 0 --- 0
D Real-Time Systems (which requires Annex C)	58	5	46	0	7
E Distributed Systems	26	0	--	--	untested
F Information Systems	21	0	--	--	untested
G Numerics	29	1	28	0	0
H Safety and Security	30	0	--	--	untested

Attachment to VC 980605E2.1-022:
Quantitative Validation Test Results

DECLARATION OF CONFORMANCE

Customer: OC Systems, Inc.

Ada Validation Facility: Electronic Data Systems
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton, OH 45424-0593
U.S.A.

ACVC Version: 2.1

Ada Implementation

Ada Compiler Name and Version: PowerAda 4.0

Host Computer System: IBM RS/6000 model 7025-F50
AIX 4.2

Target Computer System: Motorola Powerstack MT 603-66
LynxOS 2.4.0

Declaration

I, the undersigned, declare that I have no knowledge of deliberate deviations from the Ada Language Standard ANSI/ISO/IEC 8652:1995, FIPS PUB 119-1 other than the omission of features as documented in this Validation Summary Report.

_____ _____

Customer Signature Date

CHAPTER 1

INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro97] against the Ada Standard [Ada95] using the Ada Compiler Validation Capability (ACVC) Version 2.1. This Validation Summary Report (VSR) gives an account of the testing of this Ada implementation. For any technical terms used in this report, the reader is referred to [Pro97]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG97].

1.1 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the Ada Certification Body 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). Validated status is awarded only to the implementation identified in this report. Copies of this report are available to the public from the AVF that performed this validation.

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to the Ada Validation Organization. For all points of contact see Appendix B.

1.2 ACVC TEST CLASSES

Compliance of Ada implementations is tested by means of the ACVC. The ACVC contains a collection of test 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. Class A, C, D, and E tests are executable. Class B and most Class L tests are expected to produce errors at compile time and link time,respectively.

The executable tests are written in a self-checking manner and produce a PASSED, FAILED, or NOT APPLICABLE message indicating the result when they are executed. Three Ada library units, the packages REPORT and SPPRT13, and the procedure CHECK_FILE are used for this purpose. The package REPORT also provides a set of identity functions used to defeat some compiler

optimizations allowed by the Ada Standard that would circumvent a test objective. The package SPPRT13 contains constants of type SYSTEM.ADDRESS. These constants are used by selected Section 13 tests and by isolated tests for other sections. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for the Input-Output features of the Ada Standard, defined in Annex A of [Ada 95]. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. If these units are not operating correctly, validation testing is discontinued.

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 all violations of the Ada Standard are detected. Some of the Class B tests contain legal Ada code which must not be flagged illegal by the compiler. This behavior is also verified.

Class L tests check that an Ada implementation correctly detects violation of the Ada Standard involving multiple, separately compiled units. In most Class L tests, errors are expected at link time, and execution must not begin. Other L tests may execute and report the appropriate result.

For some tests of the ACVC, certain implementation-specific values must be supplied. Two insertion methods for the implementation-specific values are used: a macro substitution on the source file level of the test, and linking of a package that contains the implementation specific values. Details are described in [UG97]. A list of the values used for this implementation, along with the specification and body of the package (and children applicable to any of Specialized Needs Annexes being tested) are provided in Section 3.2 of this report.

In addition to these anticipated test modifications, changes may be required to remove unforeseen conflicts between the tests and implementation-dependent characteristics. The modifications required for this implementation are described in Section 2.2.

For the validation of each Ada implementation, a customized test suite is produced by the AVF. This customization consists of making the modifications described in the preceding paragraph, removing withdrawn tests (see Section 2.1), and possibly removing some inapplicable tests (see Section 2.1 and [UG97]).

1.3 LEGACY TESTS

ACVC 2.1 consists of legacy tests and tests specific to Ada 95. The legacy tests were taken from ACVC 1.12 with possibly minor modifications to remove incompatibilities with Ada 95. The remaining tests were developed in order to test new features of Ada 95. A consequence of this approach is that the naming conventions for tests are not uniform. The test name of a legacy test always refers to the Ada 83 Standard, even if the feature covered by the test was moved to a different section in [Ada95].

1.4 DEFINITION OF TERMS

Acceptable result	A result that is explicitly allowed by the grading criteria of the test program for a grade of passed or inapplicable.
Ada compiler	The software and any needed hardware that have to be added to a given host and target computer system to allow transformation of Ada programs into executable form and execution thereof.
Ada Compiler Validation Capability (ACVC)	The means for testing compliance of Ada implementations, consisting of the test suite, the support programs, the ACVC user's guide, and the template for the Validation Summary Report.
ACVC Maintenance Organization (AMO)	The part of the certification body that maintains the ACVC.
Ada Implementation	An Ada compilation system, including any required runtime support software, together with its host computer system and its target computer system.
Ada Joint Program Office (AJPO)	The part of the certification body which provides policy and guidance for the Ada certification system.
Ada Validation Facility (AVF)	The part of the certification body which carries out the procedures required to establish the compliance of an Ada implementation.
Ada Validation Organization (AVO)	The part of the certification body that provides technical guidance for operations of the Ada certification system.
Certification Body	The organizations (AJPO, AVO, AVFs), collectively responsible for defining and implementing Ada validation policy, including production and maintenance of the ACVC tests, and awarding of Ada validation certificates.
Compliance of an Ada Implementation	The ability of the implementation to pass an ACVC version.
Computer System	A functional unit, consisting of one or more computers and associated software, that uses common storage for all or part of a program and also for all or part of the data necessary for the execution of the program; executes user-written or user-designated programs; performs user-designated data manipulation, including arithmetic operations and logic

	operations; and that can execute programs that modify themselves during execution. A computer system may be a stand-alone unit or may consist of several inter-connected units.
Conformity	Fulfillment by a product, process or service of all requirements specified.
Customer	An individual or corporate entity who enters into an agreement with an AVF which specifies the terms and conditions for AVF services (of any kind) to be performed.
Declaration of Conformance	A formal statement from a customer assuring that conformity is realized or is attainable on the Ada implementation for which validation status is realized.
Foundation Unit (Foundation Code)	An Ada package used by multiple tests. Foundation units are designed to be reusable. A valid foundation unit must be in the Ada library for those tests that are dependent on the foundation unit.
Host Computer System	A computer system where Ada source programs are transformed into executable form.
Inapplicable Test	A test that contains one or more test objectives found to be irrelevant for the given Ada implementation.
ISO	International Organization for Standardization.
Operating System	Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management.
Specialized Needs Annex	One of annexes C through H of [Ada95]. Validation against one or more specialized needs annexes is optional. For each annex, there is a test set that applies to it. In addition to all core language tests, the appropriate set of tests must be processed satisfactorily for an implementation to be validated for a specialized needs annex.
Target Computer System	A computer system where the executable form of Ada programs are executed.
Unsupported Feature Test	A test for a language feature that is not required to be supported, because it is based upon a requirement stated in an Ada 95 Specialized Needs Annex.
Validated Ada Compiler	The compiler of a validated Ada implementation.
Validated Ada Implementation	An Ada implementation that has been validated successfully either by AVF testing or by registration [Pro97].

Validation — The process of checking the conformity of an Ada compiler to the Ada programming language and of issuing a certificate for this implementation.

Withdrawn Test — A test found to be incorrect and not used in conformity testing. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains erroneous or illegal use of the Ada programming language.

CHAPTER 2

IMPLEMENTATION DEPENDENCIES

2.1 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. Reasons for a test's inapplicability may be supported by documents issued by the ISO and the AJPO known as Ada Commentaries and commonly referenced in the format AI95-ddddd. For this implementation, the following tests were determined to be inapplicable for the reasons indicated; references to Ada Commentaries are included as appropriate.

C45322A, C45523A, and C45622A check that the proper exception is raised if MACHINE_OVERFLOWS is TRUE and the results of various floating-point operations lie outside the range of the base type; for this implementation, MACHINE_OVERFLOWS is FALSE.

C45531M..P and C45532M..P (8 tests) check fixed-point operations for types that require a SYSTEM.MAX_MANTISSA of 47 or greater; for this implementation, MAX_MANTISSA is less than 47.

C4A012B checks that the proper exception is raised when FLOAT'MACHINE_OVERFLOWS is TRUE for negative powers of 0.0; for this implementation, FLOAT'MACHINE_OVERFLOWS is FALSE.

EA3004G checks whether Pragma Inline is obeyed for a function called from within a package specification. This implementation does not obey Pragma Inline in this circumstance.

CD1009C checks whether a length clause can specify a non-default size for a floating-point type; this implementation does not support such sizes.

BD8001A, BD8002A, BD8003A, BD8004A..C (3 tests), and AD8011A use machine code insertions; this implementation provides no package MACHINE_CODE.

The tests listed in the following table check that USE_ERROR is raised if the given file operations are not supported for the given combination of mode and access method; this implementation supports these operations.

Test	File Operation	Mode	File Access Method
CE2102D	CREATE	IN_FILE	SEQUENTIAL_IO
CE2102E	CREATE	OUT_FILE	SEQUENTIAL_IO
CE2102F	CREATE	INOUT_FILE	DIRECT_IO
CE2102I	CREATE	IN_FILE	DIRECT_IO
CE2102J	CREATE	OUT_FILE	DIRECT_IO
CE2102N	OPEN	IN_FILE	SEQUENTIAL_IO
CE2102O	RESET	IN_FILE	SEQUENTIAL_IO
CE2102P	OPEN	OUT_FILE	SEQUENTIAL_IO
CE2102Q	RESET	OUT_FILE	SEQUENTIAL_IO
CE2102R	OPEN	INOUT_FILE	DIRECT_IO
CE2102S	RESET	INOUT_FILE	DIRECT_IO
CE2102T	OPEN	IN_FILE	DIRECT_IO
CE2102U	RESET	IN_FILE	DIRECT_IO
CE2102V	OPEN	OUT_FILE	DIRECT_IO
CE2102W	RESET	OUT_FILE	DIRECT_IO
CE3102E	CREATE	IN_FILE	TEXT_IO
CE3102F	RESET	Any Mode	TEXT_IO
CE3102G	DELETE	--------	TEXT_IO
CE3102I	CREATE	OUT_FILE	TEXT_IO
CE3102J	OPEN	IN_FILE	TEXT_IO
CE3102K	OPEN	OUT_FILE	TEXT_IO.

CE2203A checks that WRITE raises USE_ERROR if the capacity of an external sequential file is exceeded; this implementation cannot restrict file capacity.

CE2403A checks that WRITE raises USE_ERROR if the capacity of an external direct file is exceeded; this implementation cannot restrict file capacity.

CE3115A checks operations on text files when multiple internal files are associated with the same external file and one or more are open for writing; USE_ERROR is raised when this association is attempted.

CE3304A checks that SET_LINE_LENGTH and SET_PAGE_LENGTH raise USE_ERROR if they specify an inappropriate value for the external file; there are no inappropriate values for this implementation.

CE3413B checks that PAGE raises LAYOUT_ERROR when the value of the page number exceeds COUNT'LAST; for this implementation, the value of COUNT'LAST is greater than 150000, making the checking of this objective impractical.

CXB4001..9 (9 tests) depend on the availability of an interface to COBOL; this implementation does not support COBOL interfaces. (See section 2.2 re CXB4001.)

CXB5001..5 (5 tests) depend on the availability of an interface to Fortran; this implementation does not support Fortran interfaces.

2.2 MODIFICATIONS

In order to comply with the test objective it may be required to modify the test source code, the test processing method, or the test evaluation method. Modifications are allowable because at the time of test writing not all possible execution environments of the test and the capabilities of the compiler could be foreseen. Possible kinds of modification are:

- Test Modification: The source code of the test is changed.
  Examples for test modifications are the insertion of a pragma, the insertion of a representation clause, or the splitting of a B-test into several individual tests, if the compiler does not detect all intended errors in the original test.

- Processing Modification: The processing of the test by the Ada implementation for validation is changed.
  Examples for processing modification are the change of the compilation order for a test that consists of multiple compilations or the additional compilation of a specific support unit in the library.

- Evaluation Modification: The evaluation of a test result is changed.
  An example for evaluation modification is the grading of a test other than the output from REPORT.RESULT indicates. This may be required if the test makes assumptions about implementation features that are not supported by the implementation (e.g., the implementation of a file system on a bare target machine).

All modifications have been directed by the AVO after consulting the AVF and the customer on the technical justification of the modification.

Modifications were required for 47 tests.

The following 2 tests were split into two or more tests because this implementation did not report the violations of the Ada Standard in the way expected by the original tests.

BA3006A BA3006B

B392002, as directed by the AVO, was graded passed with the following grading modification:

> accept compiler error diagnostics for the (illegal) function declaration at line 186

The function Primitive_Of_Both_Func2, declared at line 186, violates the second sentence of ARM 3.9.3(10) as it is a primitive operation, declared in the private part, of a tagged type that is declared in the visible part, and the function doesn't override anything. This grading modification encourages this unintended illegality to be detected.

B393006 and BC51C02, as directed by the AVO, were graded passed with the following code modification:

```
for B393006, comment out lines 102 & 103;
for BC51C02, comment out line  194
```

These code modifications will remove unintended illegalities from the test programs, while retaining all intended illegalities (the check that is lost is that compilers don't wrongly treat Func as overriding in cases where it isn't--however, in these cases, it can't be legally declared for the particular checks).

B610001, as directed by the AVO, was graded passed with the following code modification:

```
comment out lines 221, 223, 225, & 228
```

These lines are ambiguous, by ARM 3.10.2(2) and 8.6(27).

C761007, as directed by the AVO, was graded passed with the following code modification:

```
replace line 376
  TCTouch.Validate( "GHGHIJ", "Asynchronously aborted operation" );
with:
  TCTouch.Validate( "GHIJ", "Asynchronously aborted operation" );
```

The original code will cause the check at line 376 to be failed because the procedures C761007_0.Finalize (@87ff) and C761007_1.Finalize (@133ff) both ensure that no duplicate characters are put into the check string. (The AVO requires this change so to retain this test for finalization, as several related test programs are withdrawn.)

B83E01C, B83E01D, and B83E01E, as directed by the AVO, were graded passed with the following grading modification:

> the intended illegalities for B83E01C at lines 172 & 177 (which are marked with "ERROR"), for B83E01D at lines 302 & 307 (which are marked with "ERROR"), and for B83E01E, file 3, at lines 56 & 61 (which are all marked with "ERROR") may be regarded instead as "optional error"s--i.e., they need not have corresponding diagnostic output from the compiler

Each of these cases is the declaration of a generic subprogram body. The only other modification that would address a compiler's failure to detect these lines as errors would be to split the test with the generic subprogram declarations removed; but this would simply duplicate the cases of the non-generic subprogram bodies, which is checked elsewhere in these test programs.

C980001, as directed by the AVO, was graded passed with the following code modification:

> comment out lines 251 & 274 (=> -- C980001_0.Hold_Up.Lock)

This modification is necessary in order to prevent the test from hanging with a queued call to the protected object C980001_0.Hold_Up.

CA2009C and CA2009F, as directed by the AVO, were graded passed with the following code modification:

> delete the control-Z character from each of the test files

BA21003, as directed by the AVO, was graded passed with the following processing modification:

> split the test file BA210031 at line 163, removing the subunit body of package Bad_Subunit from this otherwise error-free compilation; process the subunit as a separate compilation.

The Ada 95 standard 10.1(4) allows an implementation "to impose implementation-defined restrictions on compilations that contain multiple compilation_units", such as requiring all such units to be error free. Therefore, this test program might need to be given the above code modification.

BC3503A, as directed by the AVO, was graded passed with the following code modification:

> comment out lines 100, 109, & 118 (these lines are LEGAL in Ada 95)

Each of the package instantiations PS3, PR3, & PP3 is legal in Ada 95,

as the requirement for matching in Ada 95 is for the formal and actual access TYPES' (not the actual SUBtype's) designated subtypes.

BC3503C, as directed by the AVO, was graded passed with the following code modification:

comment out line 63 (this line is LEGAL in Ada 95)

The package instantiation PU3 is legal in Ada 95 (see BC3503A's entry).

CD30005, as directed by the AVO, was graded passed with the following code modification:

at lines 134 & 148 of test file cd300050, change the procedure identifier from 'CD30005' to 'CD300050'.

This change will bring the main procedure name into conformity with the ACVC main-unit naming convention (and simplify ACVC processing).

BDE0001, as directed by the AVO, was graded passed with the following grading modification:

accept compiler error diagnostics for lines 153 & 154

The implicit declaration of inherited function Func generated by the full type declaration at line 153, and the explicit declaration of Func at line 154, both violate the second sentence of ARM 3.9.3(10) in being declarations, in the private part, of a primitive function with a controlling result of a tagged type declared in the visible part, with neither function declaration overriding a function declared in the visible part. These declarations should be detected as illegalites. (These declarations require that the parent type's Func be inherited at the point of the private extension, yielding an implicitly declared Func in the visible part.)

CXA5012, as directed by the AVO, was graded passed with the following code modification:

at line 86, change '100_000' to '10_000'

This code modification is necessary for any implementation that defines type Integer to have a 16-bit range.

CXA5015, as directed by the AVO, was graded passed with the following code modification:

at line 252 change '4.1' to '4.0'

At line 255, T'Adjacent (TC_Float,TC_float) /= TC_Float may be True because the function result is given at greater precision for non-model 4.1 than the stored result.

CXA5A01 and CXA5A02, as directed by the AVO, were graded passed with the following code modifications:

```
for CXA5A01:
  append to line 248 " if New_Float'Machine_Overflows = True then"
  append to line 253 " end if;"
  append to line 264 " if Float'Machine_Overflows = True then"
  append to line 269 " end if;"

for CXA5A02:
  append to line 258 " if New_Float'Machine_Overflows = True then"
  append to line 263 " end if;"
  append to line 274 " if Float'Machine_Overflows = True then"
  append to line 279 " end if;"
```

These changes make certain conformity checks conditional upon the value of 'Machine_Overflows, as specified by [Ada95] clauses A.5.1(28,34) & G.2.4(2,4).

CXB3008, as directed by the AVO, was graded passed with the following code modifications:

```
at line 106, insert ' type acc_ptr is access IC.char_array; ' at
line 107, change function String_To_Double's parameter profile to:
'(The_String : in IC.char_array ; End_Ptr: acc_ptr := null)'

at line 125, change 'atof' to 'strtod'
```

This code modification imports the C library's strtod function, which has ANSI-defined semantics in the case of a string that doesn't conform to the model for a numeric value and so should enable the test to run as expected. (In some implementations of the C language, function atof will not return the expected value 0.0 in this case; its value is not defined.)

CXB3009, as directed by the AVO, was graded passed with the following code modification:

```
comment out lines 264..287
```

This change simply removes the entire test block beginning at line 264, which checks that Storage_Error is raised as per the standard B.3.1(28). There are many reasons why the expected Storage_Error might not be raised --too much available storage, too little time, even storage reclamation!

CXB3010, as directed by the AVO, was graded passed with the following code modification:

replicate line 199 at line 256, to update the pointer object's value:

```
TC_chars_ptr  := ICS.New_Char_Array(TC_char_array_2);
```

The change is necessary to ensure that TC_chars_ptr has a valid pointer value; the original code references TC_chars_ptr after Free was applied to it, and so by B.3.1(51,53) that execution may be erroneous.

CXB4001, as directed by the AVO, was processed with the following code modification, and was graded inapplicable (see Section 2.1):

at line 198: change 'To_Comp' to 'To_Binary'

The function To_Comp was defined in draft versions of the Ada 95 standard but was changed to To_Binary for the final (B.4:45).

CXB4007, as directed by the AVO, was processed with the following code modification, and was graded inapplicable (see Section 2.1):

comment out lines 263..268

The Byte_Array values returned by two calls of To_Binary should not be expected to be equal, contrary to this particular check.

CXB5004, as directed by the AVO, was processed with the following code modification, and was graded inapplicable (see Section 2.1):

at line f0-79, change 'INVARR(3)' to 'INVARR' [nb: not line 81]

at line f0-83, change 'STR' to 'STR *7'

The changes specified above are necessary in order to produce a legal Fortran program to be used for the test program's interfacing checks.

BXC6001, as directed by the AVO, was graded passed with the following grading modification:

accept compiler error diagnostics for the pragma Atomic at line 189

As per [Ada95] clause C.6(10), an implementation must reject the application of a pragma Atomic to any type for which it doesn't support indivisible reads or writes.

BXC6A01, BXC6A02, and BXC6A04, as directed by the AVO, were graded passed with the following code modification to the foundation file FXC6A00:

    comment out lines 103 & 113

The application of a pragma Volatile to derived types Volatile_Composite and Volatile_Array violates 13.1(10), for these types are untagged derived types (with tagged components) whose parent types are by-reference types (by 6.2:5,8). The only test that references these two types is BXC6A03, and this test is withdrawn (for a similar reason).

In addition, BXC6A02 was subjected to the following grading modification:

    accept compiler error diagnostics for lines 159 & 188

The two cited lines use an object of type FXC6A00.NonVolatile_Tagged in ways that depend its indeed being non-volatile. But this type inherits volatility from its parent.

CXD1008, as directed by the AVO, was graded passed with the following code modification:

    comment out the check @228..232

This check may fail if an implementation uses different representations (lengths) of the compared values--one possibly the register contents of evaluation, the other a stored copy--, as the value is not a model number.

CXD2001..4 and CXD2006..8 (7 tests), as directed by the AVO, were graded unsupported by the grading modification of accepting compile-time rejection of "pragma Task_Dispatching_Policy (FIFO_Within_Priorities)". (See Section 2.3.)

CXD6001, as directed by the AVO, was graded passed with the following code modifications:

     at line 114 insert 'with ImpDef;'
     at lines 270, 285, & 300 append ' Delay ImpDef.Clear_Ready_Queue;'

This delay statement will enable the Victim_Type tasks to complete before Check_Results is called.

CXD6002, as directed by the AVO, was graded passed with the following code modification (which is allowed for all implementations):

    insert immediately after line 348: CXD6002_1.Done;
    (i.e., replicate line 357 here)

On a non-uni-processor system, this code may be necessary to terminate the task CXD6002_1.Weapon (line 110).

CXD8002, as directed by the AVO, was graded passed with the following code modifications:

insert at line 169:

'Delay_Amount := RT.To.Time_Span (Duration'Small);'

and comment out line 170 ('--Delay_Amount := RT.Time_Span_Unit;')

The conversion To_Duration(Delay_Amount) at line 174 can yield 0.0, since Delay_Amount was assigned Ada.Real_Time.Time_Span_Unit, which may be smaller than Duration'Small.

at line 73, prepend 'if Available_Entry'Count > 0 then '
at line 79, append ' end if;'

These changes guard the Background_Passive task's accept statement from being executed when there is no call queued, thus preventing the task from hanging on this accept when the Background_Active task termintates (and never calls again).

CXG1004, as directed by the AVO, was graded passed with the following code modification:

at lines 294,307,320,333 replace characters '_i' with '_One'

The required change makes these assignments use the intended variable. The test was coded with a simple typographical error in what are checks of a clearly defined requirement--that Constraint_Error be raised for the complex elementary functions Arctanh & Arccoth with a parameter of plus of minus one. Implementers of the Numerics Annex should understand these requirements regardless of the coding of this ACVC test program.

CXG2002, as directed by the AVO, was graded passed with the following code modification:

at lines 99 & 279 change the expression
'Mre * abs Expected * Real'Model_Epsilon'
to: 'Mre *(abs Expected * Real'Model_Epsilon)'

This change will ensure that the expression is not evaluated by multiplying its two large terms together and overflowing.

CXG2004, as directed by the AVO, was graded passed with the following code modification:

comment out lines 455, 456, & 457 (calls to Sin_Check & Cos_Check)

By removing the calls to the flawed routines, the test program's two other, valid, routines can still be used.

CXG2008, as directed by the AVO, was graded passed with the following code modification:

for each of lines 216, 233, 489, 506, 758, 775 append (after 'begin'): ' if Real'Machine_Overflows then'

for each of lines 220, 237, 493, 510, 762, & 779 prepend (before 'exception'): 'end if; '

These code modifications correct the test program's incorrect assumption that Constraint_Error must be raised by complex division by zero, which is contrary to the allowance given by the Ada 95 standard G.1.1(40).

CXG2011, as directed by the AVO, was graded passed with the following code modification:

at line 394: change 'Failed' to 'Comment'

This change allows the non-conforming raising of Argument_Error, and so does not penalize implementers for meeting the test's original requirement. However, implementations should raise Constraint_Error in this case, as per A.5.1(28,29), which will be required under ACVC 2.2 validation.

CXG2012 and CXG2020, as directed by the AVO, were graded passed with the following code modifications:

at lines 99, 124, & 119, respectively of CXG2012 & CXG2020,
change the expression
'Mre * abs Expected * Real'Model_Epsilon'
to: 'Mre *(abs Expected * Real'Model_Epsilon)'

This change will ensure that the expression is not evaluated by multiplying its two large terms together and overflowing.

CXG2013, as directed by the AVO, was graded passed with the following code modifications:

at line 89: change '1000' to '1001'

The change should preclude an even factor of 0.5 in the expression at line 295, and hence the even results of Pi for X and Pi/2 for Y --which is sufficiently near a pole of the Tan function and may overflow (A.5.1:34).

comment out line 434 (the call to Special_Angle_Test)

By removing the call to the flawed routine, the test program's other, valid, checks can be made. (The Special_Angle_Test is argued to be too lenient, re Tan with cycle=360.0 degrees, and too severe, re cycle in radians.)

CXG2014, as directed by the AVO, was graded passed with the following code modification:

comment out line 345 (the call to Subtraction_Error_Test)

By effectively deleting this one line, the flawed subprogram will be removed from execution, and the other, valid checks can be made.

CXG2016, as directed by the AVO, was graded passed with the following code modification:

comment out lines 417 & 418

These lines contain the only calls to the incorrect procedure Indentity_1_Test. The "conversion to degrees" at line 280 is not sensible, and will wrongly cause the test to be failed.

CXG2017, as directed by the AVO, was graded passed with the following code modifications:

change line 212, by inserting parens, from

```
X :=  (B - A) * Real (I) / Real (Max_Samples) + A;
```

to

```
X :=  (B - A) *(Real (I) / Real (Max_Samples))+ A;
```

comment out line 256 (the first call to Identity_Test)

The first code modification removes the potential for overflow, forcing one of the allowed orders of evaluation for the original code. The second change removes the invocation of Identity_Test that checks Tanh values that are too close to zero for the test's error bounds.

2.3 UNSUPPORTED FEATURES OF THE ADA 95 SPECIALIZED NEEDS ANNEXES

As allowed by [Ada95], an implementation need not support any of the capabilities specified by a Specialized Needs Annex, or it may support some or all of them. For validation testing, each set of tests for a particular Annex is processed only upon customer request, but is processed in full (even if the Ada implementation provides only partial support). When such a test cannot be passed, because the implementation provides only partial support, the result is graded "unsupported" (rather than "inapplicable").

The set of tests for each of the following Specialized Needs Annexes was not processed during this validation testing:

> Annex E, Distributed Systems (all BXE* & CXE* files) Annex F, Information Systems (all BXF* & CXF* files) Annex H, Safety and Security (all BXH*, CXH*, & LXH* files)

The following tests for Annex C, Systems Programming, were graded "unsupported": none.

The following tests for Annex D, Real-Time Systems, were graded "unsupported":

> CXD2001..4 and CXD2006..8 (7 tests) check various conditions when a pragma Task_Dispatching_Policy specifies FIFO_Within_Priorities; this implementation doesn't support this dispatching policy, and so rejects the pragma at compile time. (See Section 2.2.)

The following tests for Annex G, Numerics, were graded "unsupported": none.

CHAPTER 3

PROCESSING INFORMATION

3.1 VALIDATION PROCESS

A full prevalidation was conducted at the AVF's site.

Validation testing of this Ada implementation was conducted at the customer's site by a validation team from the AVF.

A floppy diskette containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The contents of the floppy diskette were loaded directly onto the host computer.

After the test files were loaded onto the host computer, the full set of tests was processed by the Ada implementation.

The tests were compiled, linked, and executed on the host computer system.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix A for a complete listing of the processing options for this implementation. It also indicates the default options. The following explicit option settings were used for this validation:

Before initiating the processing of any Ada code, the script "setup_powerada" was executed and the target mode (LynxOS) was provided by the user.

The following option on the alibinit command is used in processing the customized test suite:

-F
   Causes the alibinit command to initialize an existing sublibrary without prompting for confirmation. This option unlocks the sublibrary if necessary before initializing it. Otherwise you would need to run the compiler with the -u option.

The following options on the ada command are used in processing the

customized test suite:

-b UnitName
   Binds and links an executable with UnitName as the main unit.

   With this option you can omit the Ada source file names on the command line. Omitting source file names causes the compiler to invoke the linkage editor to produce an executable file; all compilation units must already be compiled. If you do specify Ada source files, they will be compiled first, then UnitName will be linked as the main unit.

   UnitName must be a subprogram compilation unit in the working sublibrary, but it need not be in any of the source files when you specify both a unit name and source file names.

-i FileName
   Includes the object module or archive file named by FileName in the linking of the main program. You can use this option to specify any object modules that you need to link with your Ada program.

   By default, the compiler determines which object modules will be in the executable file based on the contents of the program library.

-l
   Creates one or more files (with the suffix .lst) containing a source listing and a list of all syntax and semantic errors the compiler finds in the Ada source code.

   One output file is created for each Ada source file specified on the command line (or on standard input if -I is used). The Ada source lines are preceded by line numbers to make it easier to locate specific places within the source files. Any errors the compiler finds are printed below the line where the error was found.

-o Name
   Specifies the name for the executable output file. The default name is a.out. This option is only valid with the -b or -m options.

-w
   Enables wide-character source encodings.

Test output, compiler and linker listings, and job logs were captured on floppy diskette and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.2 MACRO PARAMETERS AND IMPLEMENTATION-SPECIFIC VALUES

This section contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG97]. The parameter values are presented in two tables. The first table lists the values that are defined in terms of the maximum input-line length, which is the value for $MAX_IN_LEN, also listed here. These values are expressed in a symbolic notation, using placeholders as appropriate.

3.2.1 Macro Parameters

Macro Parameter	Macro Value
$MAX_IN_LEN	200
$BIG_ID1	AAA ... A1 (200 characters)
$BIG_ID2	AAA ... A2 (200 characters)
$BIG_ID3	AAA ... A3A ... A (200 characters)
$BIG_ID4	AAA ... A4A ... A (200 characters)
$BIG_STRING1	"AAA ... A" (200/2 characters)
$BIG_STRING2	"AAA ... A1" ((200/2)-1 characters)
$BLANKS	" ... " (200-20 blanks)
$MAX_STRING_LITERAL	"AAA ... A" (200 characters)
$ACC_SIZE	32
$ALIGNMENT	4
$COUNT_LAST	2_147_483_646
$ENTRY_ADDRESS	INTERRUPT0
$ENTRY_ADDRESS1	INTERRUPT1
$ENTRY_ADDRESS2	INTERRUPT2
$FIELD_LAST	1000
$FORM_STRING	""
$FORM_STRING2	"CANNOT_RESTRICT_FILE_CAPACITY"
$GREATER_THAN_DURATION	100_000.0

PROCESSING INFORMATION

$ILLEGAL_EXTERNAL_FILE_NAME1	" & ASCII.NUL & "
$ILLEGAL_EXTERNAL_FILE_NAME2	" & ASCII.NUL & ASCII.NUL & "
$INAPPROPRIATE_LINE_LENGTH	-1
$INAPPROPRIATE_PAGE_LENGTH	-1
$INTEGER_FIRST	-2147483648
$INTEGER_LAST	2147483647
$LESS_THAN_DURATION	-100_000.0
$MACHINE_CODE_STATEMENT	NULL;
$MAX_INT	2147483647
$MIN_INT	-2147483648
$NAME	SHORT_SHORT_INTEGER
$NAME_SPECIFICATION1	/ce/X2120A
$NAME_SPECIFICATION2	/ce/X2120B
$NAME_SPECIFICATION3	/ce/X3119A
$OPTIONAL_DISC	NO_SUCH_MACHINE_CODE_DISC
$RECORD_DEFINITION	RECORD NULL; END RECORD;
$RECORD_NAME	NO_SUCH_MACHINE_CODE_TYPE
$TASK_SIZE	32
$TASK_STORAGE_SIZE	32768
$VARIABLE_ADDRESS	VARIABLE0'ADDRESS
$VARIABLE_ADDRESS1	VARIABLE1'ADDRESS
$VARIABLE_ADDRESS2	VARIABLE2'ADDRESS

Package ImpDef and Its Children

The package ImpDef is used by several tests of core language features. Before use in ACVC testing, this package is modified to specify certain implementation-defined features. In addition, package ImpDef has a child package for each Specialized Needs Annex, each of which may need similar modifications. The child packages are independent of one another, and are used only by tests for their respective annexes.

This section presents the package ImpDef and each of the relevant child packages as they were modified for this validation. In the interests of simplifying this VSR, the header comment block was removed from each of the package files.

3.2.1.1 Package ImpDef

```
with Report;
with Ada.Text_IO;
with System.Storage_Elements;
with Interfaces.C.Strings;

package ImpDef is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following boolean constants indicate whether this validation will
   -- include any of annexes C-H. The values of these booleans affect the
   -- behavior of the test result reporting software.
   --
   --    True  means the associated annex IS included in the validation.
   --    False means the associated annex is NOT included.

   Validating_Annex_C : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_D : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_E : constant Boolean := False;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_F : constant Boolean := False;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_G : constant Boolean := True;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

   Validating_Annex_H : constant Boolean := False;
   --                                       ^^^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This is the minimum time required to allow another task to get
```

```
   -- control.  It is expected that the task is on the Ready queue.
   -- A duration of 0.0 would normally be sufficient but some number
   -- greater than that is expected.

   Minimum_Task_Switch : constant Duration := 0.1;
   --                                         ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This is the time required to activate another task and allow it
   -- to run to its first accept statement.  We are considering a simple task
   -- with very few Ada statements before the accept.  An implementation is
   -- free to specify a delay of several seconds, or even minutes if need be.
   -- The main effect of specifying a longer delay than necessary will be an
   -- extension of the time needed to run the associated tests.

   Switch_To_New_Task : constant Duration := 1.0;
   --                                       ^^^ -- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This is the time which will clear the queues of other tasks
   -- waiting to run.  It is expected that this will be about five
   -- times greater than Switch_To_New_Task.

   Clear_Ready_Queue : constant Duration := 5.0;
   --                                     ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Some implementations will boot with the time set to 1901/1/1/0.0
   -- When a delay of Delay_For_Time_Past is given, the implementation
   -- guarantees that a subsequent call to Ada.Calendar.Time_Of(1901,1,1)
   -- will yield a time that has already passed (for example, when used in
   -- a delay_until statement).

   Delay_For_Time_Past : constant Duration := 0.1;
   --                                         ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Minimum time interval between calls to the time dependent Reset
   -- procedures in Float_Random and Discrete_Random packages that is
   -- guaranteed to initiate different sequences.  See RM A.5.2(45).

   Time_Dependent_Reset : constant Duration := 0.3;
   --                                          ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Test CXA5013 will loop, trying to generate the required sequence
   -- of random numbers.  If the RNG is faulty, the required sequence
   -- will never be generated.  Delay_Per_Random_Test is a time-out value
   -- which allows the test to run for a period of time after which the
```

```
   -- test is failed if the required sequence has not been produced.
   -- This value should be the time allowed for the test to run before it
   -- times out.  It should be long enough to allow multiple (independent)
   -- runs of the testing code, each generating up to 1000 random
   -- numbers.

   Delay_Per_Random_Test : constant Duration := 1.0;
   --                                           ^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The time required to execute this procedure must be greater than the
   -- time slice unit on implementations which use time slicing.  For
   -- implementations which do not use time slicing the body can be null.

   procedure Exceed_Time_Slice;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This constant must not depict a random number generator state value.
   -- Using this string in a call to function Value from either the
   -- Discrete_Random or Float_Random packages will result in
   -- Constraint_Error (expected result in test CXA5012).

   Non_State_String : constant String := "By No Means A State";
   --           MODIFY HERE AS NEEDED --- ^^^^^^^^^^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This string constant must be a legal external tag value as used by
   -- CD10001 for the type Some_Tagged_Type in the representation
   -- specification for the value of 'External_Tag.

   External_Tag_Value : constant String := "implementation_defined";
   --              MODIFY HERE AS NEEDED --- ^^^^^^^^^^^^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following address constant must be a valid address to locate
   -- the C program CD30005_1.  It is shown here as a named number;
   -- the implementation may choose to type the constant as appropriate.

   CD30005_1_Foreign_Address : constant System.Address:=
                  system.label("_cd30005_1");
   --             System.Storage_Elements.To_Address ( 16#0000_0000# );
   --                      MODIFY HERE AS REQUIRED --- ^^^^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constant must be the external name resulting
   -- from the C compilation of CD30005_1.  The string will be used as an
   -- argument to pragma Import.

   CD30005_1_External_Name : constant String := "_cd30005_1";
```

```
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following constants should represent the largest default alignment
   -- value and the largest alignment value supported by the linker.
   -- See RM 13.3(35).

   Max_Default_Alignment : constant := 8;
   --                                  ^ --- MODIFY HERE AS NEEDED

   Max_Linker_Alignment  : constant := 8;
   --                                  ^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constants must be the external names resulting
   -- from the C compilation of CXB30130.C and CXB30131.C.  The strings
   -- will be used as arguments to pragma Import.

   CXB30130_External_Name : constant String := "CXB30130";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB30131_External_Name : constant String := "CXB30131";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constants must be the external names resulting
   -- from the COBOL compilation of CXB40090.CBL, CXB40091.CBL, and
   -- CXB40092.CBL.  The strings will be used as arguments to pragma Import.

   CXB40090_External_Name : constant String := "CXB40090";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB40091_External_Name : constant String := "CXB40091";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB40092_External_Name : constant String := "CXB40092";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constants must be the external names resulting
   -- from the Fortran compilation of CXB50040.FTN, CXB50041.FTN,
   -- CXB50050.FTN, and CXB50051.FTN.
   --
   -- The strings will be used as arguments to pragma Import.
   --
   -- Note that the use of these four string constants will be split between
   -- two tests, CXB5004 and CXB5005.

   CXB50040_External_Name : constant String := "CXB50040";
   --                  MODIFY HERE AS NEEDED --- ^^^^^^^^
```

```
   CXB50041_External_Name : constant String := "CXB50041";
   --                 MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50050_External_Name : constant String := "CXB50050";
   --                 MODIFY HERE AS NEEDED --- ^^^^^^^^

   CXB50051_External_Name : constant String := "CXB50051";
   --                 MODIFY HERE AS NEEDED --- ^^^^^^^^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following constants have been defined for use with the
   -- representation clause in FXACA00 of type Sales_Record_Type.
   --
   -- Char_Bits should be an integer at least as large as the number
   -- of bits needed to hold a character in an array.
   -- A value of 6 * Char_Bits will be used in a representation clause
   -- to reserve space for a six character string.
   --
   -- Next_Storage_Slot should indicate the next storage unit in the record
   -- representation clause that does not overlap the storage designated for
   -- the six character string.

   Char_Bits         : constant := 8;
   --     MODIFY HERE AS NEEDED ---^

   Next_Storage_Slot : constant := 6;
   --     MODIFY HERE AS NEEDED ---^

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following string constant must be the path name for the .AW
   -- files that will be processed by the Wide Character processor to
   -- create the C250001 and C250002 tests.  The Wide Character processor
   -- will expect to find the files to process at this location.

   Test_Path_Root : constant String :=
      "/build10/validation_pa4/acvc-ocs.lynx/c2/";
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ --- MODIFY HERE AS NEEDED

   -- The following two strings must not be modified unless the .AW file
   -- names have been changed.  The Wide Character processor will use
   -- these strings to find the .AW files used in creating the C250001
   -- and C250002 tests.

  Wide_Character_Test : constant String := Test_Path_Root & "c250001";
  Upper_Latin_Test    : constant String := Test_Path_Root & "c250002";

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The following instance of Integer_IO or Modular_IO must be supplied
   -- in order for test CD72A02 to compile correctly.
   -- Depending on the choice of base type used for the type
```

```
   -- System.Storage_Elements.Integer_Address; one of the two instances will
   -- be correct.  Comment out the incorrect instance.

   --I package Address_Value_IO is
   --I  new Ada.Text_IO.Integer_IO(System.Storage_Elements.Integer_Address);

       package Address_Value_IO is
        new Ada.Text_IO.Modular_IO(System.Storage_Elements.Integer_Address);

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef;


     --=================================================================--


package body ImpDef is

   -- NOTE: These are example bodies.  It is expected that implementors
   --       will write their own versions of these routines.

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The time required to execute this procedure must be greater than the
   -- time slice unit on implementations which use time slicing.  For
   -- implementations which do not use time slicing the body can be null.

   Procedure Exceed_Time_Slice is
      T : Integer := 0;
      Loop_Max : constant Integer := 4_000;
   begin
      for I in 1..Loop_Max loop
         T := Report.Ident_Int (1) * Report.Ident_Int (2);
      end loop;
   end Exceed_Time_Slice;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef;
```

3.2.1.2 Package ImpDef.Annex_C

```
with Ada.Interrupts.Names;

package ImpDef.Annex_C is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Interrupt_To_Generate should identify a non-reserved interrupt
   -- that can be predictably generated within a reasonable time interval
   -- (as specified by the constant Wait_For_Interrupt) during testing.

   Interrupt_To_Generate: constant Ada.Interrupts.Interrupt_ID :=
--       Ada.Interrupts.Interrupt_ID'First;  -- to allow trivial compilation
         Ada.Interrupts.Names.SIGUSR1;
   -- ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Wait_For_Interrupt should specify the reasonable time interval during
   -- which the interrupt identified by Interrupt_To_Generate can be
   -- expected to be generated.

   Wait_For_Interrupt : constant := 10.0;
   --                              ^^^^ --- MODIFY HERE AS NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Enable_Interrupts should enable interrupts, if this
   -- is required by the implementation. [See additional notes on this
   -- procedure in the package body.]

   procedure Enable_Interrupts;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Generate_Interrupt should generate the interrupt
   -- identified by Interrupt_To_Generate within the time interval
   -- specified by Wait_For_Interrupt. [See additional notes on this
   -- procedure in the package body.]

   procedure Generate_Interrupt;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_C;
```

PROCESSING INFORMATION

```
     --=====================================================================--


package body ImpDef.Annex_C is

   -- NOTE: These are example bodies.  It is expected that implementors
   --       will write their own versions of these routines.

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Enable_Interrupts should enable interrupts, if this
   -- is required by the implementation.
   --
   -- The default body is null, since it is expected that most implementations
   -- will not need to perform this step.
   --
   -- Note that Enable_Interrupts will be called only once per test.

   procedure Enable_Interrupts is
   begin
      null;

   -- ^^^^^^^^^^^^^^^^^^^  MODIFY THIS BODY AS NEEDED  ^^^^^^^^^^^^^^^^^^^

   end Enable_Interrupts;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- The procedure Generate_Interrupt should generate the interrupt
   -- identified by Interrupt_To_Generate within the time interval
   -- specified by Wait_For_Interrupt.
   --
   -- The default body assumes that an interrupt will be generated by some
   -- physical act during testing. While this approach is acceptable, the
   -- interrupt should ideally be generated by appropriate code in the
   -- procedure body.
   --
   -- Note that Generate_Interrupt may be called multiple times by a single
   -- test. The code used to implement this procedure should account for this
   -- possibility.

   procedure Generate_Interrupt is

     function Getpid return Integer;
       pragma Import(C, Getpid);

     procedure Kill(Pid : Integer; Signal : Integer);
       pragma Import(C, Kill);
```

```
   begin
      Report.Comment (". >>>>> GENERATE THE INTERRUPT NOW <<<<< ");
      Kill(Getpid,Integer(Interrupt_To_Generate));

   -- ^^^^^^^^^^^^^^^^^^^  MODIFY THIS BODY AS NEEDED  ^^^^^^^^^^^^^^^^^^^

   end Generate_Interrupt;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_C;
```

3.2.1.3 Package ImpDef.Annex_D

```
package ImpDef.Annex_D is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This constant is the maximum storage size that can be specified
   -- for a task.  A single task that has this size must be able to
   -- run.  Ideally, this value is large enough that two tasks of this
   -- size cannot run at the same time.  If the value is too small then
   -- test CXDC001 may take longer to run.  See the test for further
   -- information.

   Maximum_Task_Storage_Size : constant := 16_000_000;
   --                                      ^^^^^^^^^^ --- MODIFY HERE AS
NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- Indicates the type of processor on which the tests are running.
   -- Time_Slice indicates a uniprocessor with an operating system that
   -- simulates a multi-processor by using time slicing.

   type Processor_Type is (Uni_Processor, Time_Slice, Multi_Processor);

   Processor : constant Processor_Type := Uni_Processor;
   --                                     ^^^^^^^^^^ --- MODIFY HERE AS
NEEDED

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_D;
```

3.2.1.4 Package ImpDef.Annex_G

```
package ImpDef.Annex_G is

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This function must return a "negative zero" value for implementations
   -- for which Float'Signed_Zeros is True.

   function Negative_Zero return Float;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_G;


     --=================================================================--


package body ImpDef.Annex_G is

   -- NOTE: These are example bodies.  It is expected that implementors
   --       will write their own versions of these routines.

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

   -- This function must return a negative zero value for implementations
   -- for which Float'Signed_Zeros is True.
   --
   -- The default body simply returns a negated literal 0.0.  If the
   -- default body does not return the value corresponding to a negatively
   -- signed zero for the implementation under test, it must be replaced
   -- by one which does.  See RM A.5.3(13).

   function Negative_Zero return Float is
   begin
      return -0.0;      -- Note: If this value is not negative zero for the
                        --       implementation, use of this "default" value
                        --       could result in false failures in
                        --       implementations where Float'Signed_Zeros
                        --       is True.

   -- ^^^^^^^^^^^^^^^^^^^  MODIFY THIS BODY AS NEEDED  ^^^^^^^^^^^^^^^^^^^

   end Negative_Zero;

--=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====-=====--

end ImpDef.Annex_G;
```

3.3 WITHDRAWN TESTS

At the time of this validation testing, the following 24 tests were withdrawn from the ACVC 2.1 test suite.

B37312B	BXC6A03	C390010	C392010	C392012	C42006A
C48009A	C760007	C760012	C761006	C761008	C761009
C9A005A	C9A008A	CD20001	CXC3004	CXD2005	CXD4009
CXD5002	CXDB005	CXDC001	CXG2022	E28002B	LA1001F

APPENDIX A

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

A.1 Compilation System Options

The compiler options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report.

OPTION SETTINGS

The following options are used on the alibinit command:

-F
   Causes the alibinit command to initialize an existing sublibrary without prompting for confirmation. This option unlocks the sublibrary if necessary before initializing it. Otherwise you would need to run the compiler with the -u option.

-L LibraryList
   Uses the first sublibrary name in the file LibraryList as the name of the sublibrary to create. If you do not specify a library list file, the first sublibrary name in the default file alib.list is used.

The following options apply to the ada command:

-b UnitName
   Binds and links an executable with UnitName as the main unit.

   With this option you can omit the Ada source file names on the command line. Omitting source file names causes the compiler to invoke the linkage editor to produce an executable file; all compilation units must already be compiled. If you do specify Ada source files, they will be compiled first, then UnitName will be linked as the main unit.

UnitName must be a subprogram compilation unit in the working sublibrary, but it need not be in any of the source files when you specify both a unit name and source file names.

-i FileName
Includes the object module or archive file named by FileName in the linking of the main program. You can use this option to specify any object modules that you need to link with your Ada program.

By default, the compiler determines which object modules will be in the executable file based on the contents of the program library.

-l
Creates one or more files (with the suffix .lst) containing a source listing and a list of all syntax and semantic errors the compiler finds in the Ada source code.

One output file is created for each Ada source file specified on the command line (or on standard input if -I is used). The Ada source lines are preceded by line numbers to make it easier to locate specific places within the source files. Any errors the compiler finds are printed below the line where the error was found.

-o Name
Specifies the name for the executable output file. The default name is a.out. This option is only valid with the -b or -m options.

-w
Enables wide-character source encodings.

-k
Does not store symbolic debugging information in the working sublibrary. This information makes it possible for the debugger and other debugging tools to associate abstractions like line numbers and variable names with locations within a running program. By default, debugging information is saved in the working sublibrary; this option ensures that only information necessary to perform further compilations is saved.

Note: the addition of debugging information may increase the size of a sublibrary by as much as two and a half times. However, this information is necessary to use any PowerAda library-based debugging and development tools (e.g., "powerada").

-L Library
Specifies the name for a library list file or the root sublibrary (within a project). This determines the library where the compiler searches for compilation units and dependency information. If this option is not specified, ada first looks for the library list file alib.list in the current directory. If this is not found, and the current directory is within a project, it determines the li-

brary from the root sublibrary adalib in the current directory.

-q task_stack={ DefaultSize | environment_variable }
Specifies the default stack size for tasks, in bytes. By default, task stacks are at least 32M bytes in size. If the value specified is environment_variable, the value of the environment variable Ada_task_stack will be used (if defined) upon program startup. Note that this option does not affect the stack size for tasks whose storage size is explicitly provided in a pragma STORAGE_SIZE for the corresponding task type.

-q stack_limit={ MaximumSize | environment_variable }
Specifies the maximum size for the main program stack, in bytes, to a maximum of 256_000_000. By default, no limit is imposed by the Ada runtime. If the value specified is environment_variable, the value of the environment variable Ada_stack_limit is used (if defined). Note that in all cases, the limit cannot exceed the limit imposed by LynxOS for the process.

-q stack_heap={ MaximumSize | environment_variable }
Specifies the size, in bytes, of the storage heap for allocating task stacks. This heap is only used in programs that are bound with the LynxOS thread runtime to allocate stacks for tasks which do not execute in separate LynxOS threads. By default, the stack heap is at least 32M bytes in size. If the value specified is environment_variable, the value of the environment variable Ada_stack_heap is used (if defined).

-q heap_limit={ MaximumSize | environment_variable }
Specifies the maximum heap size for the main program, in bytes, to a maximum value of 256_000_000. By default, the size is limited only by LynxOS for the process. If the value specified is environment_variable, the value of the environment variable Ada_heap_limit is used.

-q arch={ com | pwr | pwr2 | ppc }
Specifies the architecture on which the program will be run. The options are interpreted as follows:

com - Produce an object that contains instructions that will run on all the POWER and PowerPC hardware platforms.

pwr - Produce an object that contains instructions that will run on the POWER hardware platform.

pwr2 - Produce an object that contains instructions that will run on the POWER2 hardware platform.

ppc - Produce an object that contains instructions that will run on any of the 32-bit PowerPC hardware platforms.

The default is -q arch=ppc.

-! : Enable debugging

COMPILATION SYSTEM OPTIONS AND LINKER OPTIONS

```
  -a         : produce assembler code
  -e         : do not produce an executable file
  -h         : display information about the ada command
  -ildOption : pass arguments to LynxOS linker (ld)
  -I         : read a list of files to compile from standard input
  -O         : produce optimized code
  -m         : compiles Ada source files then binds and links last unit as
               main program
  -p         : turn on execution profiling option
  -P         : do NOT copy (proprietary) source into sublibrary
  -q Parameter=Value : special parameters
     crt=FilePathName
               : specify alternative to /usr/lib/crt0.o for use when linking the
                 Ada load module
     crt_r=FilePathName
               : specify alternative to /usr/lib/crt0_r.o for use when linking
                 the Ada load module
     link_script_name=FilePathName
               : save script that executes 'ld' command to designated file
     name_after_code
               : places ASCII string of link name of subprogram after code for
                 subprogram for low-level debugging
     XlCbind : includes archive and object files necessary for binding C ++
               compiled with IBM's XlC compiler
  -qcomment=File
               : stores text from specified file as comments inside object
module
  -r UnitName: (re)instantiate implicit instance subunit
  -s         : bind a main program as a shared module
  -S         : suppress run-time checks for Ada exceptions
  -u         : unlock the working sublibrary
  -v         : verbose - display informative messages
  -W         : Suppress pragma inline warnings
  -x Name    : executable file name to rebind
  -. Option  : Pass Option to the Ada compiler
```

A.2 Linker Options

The linker options of this Ada implementation, as described in this Appendix, are provided by the customer. Unless specifically noted otherwise, references in this Appendix are to linker documentation and not to this report.

Link-time options are given with the ada command, as described in the previous Appendix.

APPENDIX B

POINTS OF CONTACT

Ada Validation Facility

Phil Brashear, AVF Manager
Electronic Data Systems
4646 Needmore Road, Bin 46
P.O. Box 24593
Dayton, OH 45424-0593
U.S.A.
Phone : (937) 237-4510
Internet : brashp@dysmailpo.deisoh.msd.eds.com

Ada Validation Organization

Mr. Clyde Roby
Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria VA 22311
U.S.A.
Phone : (703) 845-6666
FAX : (703) 345-6848
Internet : avo@sw-eng.falls-church.va.us

Ada Joint Program Office

Joan McGarity
Center for Software
Defense Information Systems Agency
5600 Columbia Pike
Falls Church VA 22041
U.S.A.
Phone : (703) 681-2453
Internet: mcgaritj@ncr.disa.mil

POINTS OF CONTACT

For technical and sales information about this Ada implementation, contact:

Tom Fleck
OC Systems, Inc.
9990 Lee Highway, Suite 270
Fairfax, VA 22030
(703) 359-8160

APPENDIX C

REFERENCES

[Ada95] Reference Manual for the Ada Programming Language, ANSI/ISO/IEC 8652:1995

[Pro97] Ada Compiler Validation Procedures, Version 5.0, Ada Validation Organization and Ada Joint Program Office, March 1997

[UG97] The Ada Compiler Validation Capability Version 2.1 User's Guide, Revision 1, SAIC and CTA, March 1997

REFERENCES

end of document

(REMOVE THIS PAGE)