AD-A240 610

# NTATION PAGE

Form Approved
OPM No. 0704-0188

(2)

1 hour per response, including the time for reviewing instructions, searching existing data sources gathering and maintaining the data this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington 5 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Information and Regulatory Affairs, Office of

1. AGENCY USE ONLY (Leave blank)

PORT DATE

3. REPORT TYPE AND DATES COVERED
Final: 02 May 1991 to 01 Jun 1993

4. TITLE AND SUBTITLE
Ada Compiler Validation Summary Report:DDC International A/S, DACS VAX/VMS to 80860 Bare Ada Cross Compiler System, Version 4.6.1, VAX 8530 (Host) to TADPOLE TECHNOLOGY PLC TP860M (Bare Board)(Target), 910502S1.11158

5. FUNDING NUMBERS

6. AUTHOR(S)
National Institute of Standards and Technology
Gaithersburg, MD
USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
National Institute of Standards and Technology
National Computer Systems Laboratory
Bldg. 255, Rm A266
Gaithersburg, MD 20899 USA

8. PERFORMING ORGANIZATION REPORT NUMBER
NIST90DDC500_10_1.11

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Ada Joint Program Office
United States Department of Defense
Pentagon, RM 3E114
Washington, D.C. 20301-3081

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
DDC International A/S, DACS VAX/VMS to 80860 Bare Ada Cross Compiler System, Version 4.6.1, Gaithersburg, MD, VAX 8530 running VMS Version 5.3 (Host) to TADPOLE TECHNOLOGY PLC TP860M (Bare Board)(Target), ACVC 1.11.



91-11051

14. SUBJECT TERMS
Ada programming language, Ada Compiler Val. Summary Report, Ada Compiler Val. Capability, Val. Testing, Ada Val. Office, Ada Val. Facility, ANSI/MIL-STD-1815A, AJPO.

15. NUMBER OF PAGES

16. PRICE CODE

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| UNCLASSIFIED | UNCLASSIFED | UNCLASSIFIED | |

AVF Control Number: NIST90DDC500_10_1.11
DATE COMPLETED
BEFORE ON-SITE: 1991-02-05
AFTER ON-SITE: 1991-05-02
REVISIONS: 1991-07-24

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 910502S1.11158
DDC International A/S
DACS VAX/VMS to 80860 Bare Ada Cross Compiler System,
Version 4.6.1
VAX 8530 => TADPOLE TECHNOLOGY PLC TP860M (Bare Board)

Prepared By:
Software Standards Validation Group
National Computer Systems Laboratory
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, Maryland 20899

AVF Control Number: NIST90DDC500_10_1.11

Certificate Information

The following Ada implementation was tested and determined to pass ACVC 1.11. Testing was completed on May 2, 1991.

| | |
|---|---|
| Compiler Name and Version: | DACS VAX/VMS to 80860 Bare Ada Cross Compiler System, Version 4.6.1 |
| Host Computer System: | VAX 8530 running VMS Version 5.3 |
| Target Computer System: | TADPOLE TECHNOLOGY PLC TP860M (Bare Board) |

A more detailed description of this Ada implementation is found in section 3.1 of this report.

As a result of this validation effort, Validation Certificate 910502S1.11158 is awarded to DDC International A/S. This certificate expires on March 01, 1993.

This report has been reviewed and is approved.

Ada Validation Facility
Dr. David K. Jefferson
Chief, Information Systems
Engineering Division (ISED)

Ada Validation Facility
Mr. L. Arnold Johnson
Manager, Software Standards
Validation Group

Computer Systems Laboratory (CLS)
National Institute of Standards and Technology
Building 225, Room A266
Gaithersburg, MD 20899

Ada Validation Organization
for Director, Computer & Software
Engineering Division
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director
Department of Defense
Washington DC 20301

DECLARATION OF CONFORMANCE

The following declaration of conformance was supplied by the customer.

DECLARATION OF CONFORMANCE

Customer and Certificate Awardee: DDC International A/S

Ada Validation Facility: National Institute of Standards and Technology
Computer Systems Laboratory (CSL)
Software Validation Group
Building 225, Room A266
Gaithersburg, Maryland 20899

ACVC Version: 1.11

Ada Implementation:

Compiler Name and Version: DACS VAX/VMS to 80860 Bare Ada Cross Compiler System, Version 4.6.1

Host Computer System: VAX 8530 running VMS Version 5.3

Target Computer System: TADPOLE TECHNOLOGY PLC TP860M (Bare Board)

Declaration:

I the undersigned, declare that I have no knowledge of deliberate deviations from the Ada Language Standard ANSI/MIL-STD-1815A ISO 8652-1987 in the implementation listed above.

Ole Dommergård
Customer Signature

1991-05-31
Date

Company DDC International A/S
Title Project Manager

TABLE OF CONTENTS


CHAPTER 1 . . . . . 1-1
INTRODUCTION . . . . . 1-1
1.1 USE OF THIS VALIDATION SUMMARY REPORT . . . . 1-1
1.2 REFERENCES . . . . . 1-1
1.3 ACVC TEST CLASSES . . . . . 1-2
1.4 DEFINITION OF TERMS . . . . . 1-3

CHAPTER 2 . . . . . 2-1
IMPLEMENTATION DEPENDENCIES . . . . . 2-1
2.1 WITHDRAWN TESTS . . . . . 2-1
2.2 INAPPLICABLE TESTS . . . . . 2-1
2.3 TEST MODIFICATIONS . . . . . 2-4

CHAPTER 3 . . . . . 3-1
PROCESSING INFORMATION . . . . . 3-1
3.1 TESTING ENVIRONMENT . . . . . 3-1
3.2 SUMMARY OF TEST RESULTS . . . . . 3-2
3.3 TEST EXECUTION . . . . . 3-3

APPENDIX A . . . . . A-1
MACRO PARAMETERS . . . . . A-1

APPENDIX B . . . . . B-1
COMPILATION SYSTEM OPTIONS . . . . . B-1
LINKER OPTIONS . . . . . B-2

APPENDIX C . . . . . C-1
APPENDIX F OF THE Ada STANDARD . . . . . C-1

# CHAPTER 1

# INTRODUCTION

The Ada implementation described above was tested according to the Ada Validation Procedures [Pro90] against the Ada Standard [Ada83] using the current Ada Compiler Validation Capability (ACVC). 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 [Pro90]. A detailed description of the ACVC may be found in the current ACVC User's Guide [UG89].

## 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). 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 implementation has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from the AVF which performed this validation or from:

National Technical Information Service
5285 Port Royal Road
Springfield VA 22161

Questions regarding this report or the validation test results should be directed to the AVF which performed this validation or to:

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

## 1.2 REFERENCES

[Ada83] Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

[Pro90] Ada Compiler Validation Procedures, Version 2.1, Ada Joint Program Office, August 1990.

[UG89] Ada Compiler Validation Capability User's Guide, 21 June 1989.

## 1.3 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 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 is used by many tests for Chapter 13 of the Ada Standard. 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 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. Errors are expected at link time, and execution is attempted.

In some tests of the ACVC, certain macro strings have to be replaced by implementation-specific values -- for example, the largest integer. A list of the values used for this implementation is provided in Appendix A. In addition to these anticipated test modifications, additional 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 2.3.
For 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 some inapplicable tests (see Section 3.2 and [UG89]).

In order to pass an ACVC an Ada implementation must process each test of the customized test suite according to the Ada Standard.

## 1.4 DEFINITION OF TERMS

| | |
|---|---|
| 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, Validation consisting of the test suite, the support programs, the ACVC Capability user's guide and the template for the validation summary (ACVC) report. |
| Ada Implementation | An Ada compiler with its host computer system and its target computer 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. |
| 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 attainable on the Ada implementation for which validation status is realized. |
| 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. |
| Operating System | Software that controls the execution of programs and that provides services such as resource allocation, scheduling, input/output control, and data management. Usually, operating systems are predominantly software, but partial or complete hardware implementations are possible. |
| Target Computer System | A computer system where the executable form of Ada programs are executed. |
| 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 [Pro90]. |
| 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 WITHDRAWN TESTS

Some tests are withdrawn by the AVO from the ACVC because they do not conform to the Ada Standard. The following 93 tests had been withdrawn by the Ada Validation Organization (AVO) at the time of validation testing. The rationale for withdrawing each test is available from either the AVO or the AVF. The publication date for this list of withdrawn tests is 91-03-14.

| | | | | | |
|---|---|---|---|---|---|
| E28005C | B28006C | C34006D | C35508I | C35508J | C35508M |
| C35508N | C35702A | C35702B | B41308B | C43004A | C45114A |
| C45346A | C45612A | C45612B | C45612C | C45651A | C46022A |
| B49008A | A74006A | C74308A | B83022B | B83022H | B83025B |
| B83025D | B83026B | C83026A | C83041A | B85001L | C86001F |
| C94021A | C97116A | C98003B | BA2011A | CB7001A | CB7001B |
| CB7004A | CC1223A | BC1226A | CC1226B | BC3009B | BD1B02B |
| BD1B06A | AD1B08A | BD2A02A | CD2A21E | CD2A23E | CD2A32A |
| CD2A41A | CD2A41E | CD2A87A | CD2B15C | BD3006A | BD4008A |
| CD4022A | CD4022D | CD4024B | CD4024C | CD4024D | CD4031A |
| CD4051D | CD5111A | CD7004C | ED7005D | CD7005E | AD7006A |
| CD7006E | AD7201A | AD7201E | CD7204B | AD7206A | BD8002A |
| BD8004C | CD9005A | CD9005B | CDA201E | CE2107I | CE2117A |
| CE2117B | CE2119B | CE2205B | CE2405A | CE3111C | CE3116A |
| CE3118A | CE3411B | CE3412B | CE3607B | CE3607C | CE3607D |
| CE3812A | CE3814A | CE3902B | | | |

## 2.2 INAPPLICABLE TESTS

A test is inapplicable if it contains test objectives which are irrelevant for a given Ada implementation. The inapplicability criteria for some tests are explained in documents issued by ISO and the AJPO known as Ada Issues and commonly referenced in the format AI-dddd. For this implementation, the following tests were inapplicable for the reasons indicated; references to Ada Issues are included as appropriate.

The following 201 tests have floating-point type declarations requiring more digits than SYSTEM.MAX_DIGITS:

| | |
|---|---|
| C24113L..Y (14 tests) | C35705L..Y (14 tests) |
| C35706L..Y (14 tests) | C35707L..Y (14 tests) |
| C35708L..Y (14 tests) | C35802L..Z (15 tests) |
| C45241L..Y (14 tests) | C45321L..Y (14 tests) |

| | |
|---|---|
| C45421L..Y (14 tests) | C45521L..Z (15 tests) |
| C45524L..Z (15 tests) | C45621L..Z (15 tests) |
| C45641L..Y (14 tests) | C46012L..Z (15 tests) |

C24113I..K (3 TESTS) USE A LINE LENGTH IN THE INPUT FILE WHICH EXCEEDS 132 CHARACTERS.

THE FOLLOWING 20 TESTS CHECK FOR THE PREDEFINED TYPE LONG_INTEGER:

| | | | | |
|---|---|---|---|---|
| C35404C | C45231C | C45304C | C45411C | C45412C |
| C45502C | C45503C | C45504C | C45504F | C45611C |
| C45613C | C45614C | C45631C | C45632C | B52004D |
| C55B07A | B55B09C | B86001W | C86006C | CD7101F |

C35404D, C45231D, B86001X, C86006E, AND CD7101G CHECK FOR A PREDEFINED INTEGER TYPE WITH A NAME OTHER THAN INTEGER, LONG_INTEGER, OR SHORT_INTEGER.

C35713B, C45423B, B86001T, AND C86006H CHECK FOR THE PREDEFINED TYPE SHORT_FLOAT.

C35713D AND B86001Z CHECK FOR A PREDEFINED FLOATING-POINT TYPE WITH A NAME OTHER THAN FLOAT, LONG_FLOAT, OR SHORT_FLOAT.

C45531M, C45531N, C45531O, C45531P, C45532M, C45532N, C45532O, AND C45532P CHECK FIXED-POINT OPERATIONS FOR TYPES THAT REQUIRE A SYSTEM.MAX_MANTISSA OF 47 OR GREATER.

C45624A AND C45624B CHECK THAT THE PROPER EXCEPTION IS RAISED IF MACHINE_OVERFLOWS IS FALSE FOR FLOATING-POINT TYPES; FOR THIS IMPLEMENTATION, MACHINE_OVERFLOWS IS TRUE.

C4A013B CONTAINS THE EVALUATION OF AN EXPRESSION INVOLVING 'MACHINE_RADIX APPLIED TO THE MOST PRECISE FLOATING-POINT TYPE. THIS EXPRESSION WOULD RAISE AN EXCEPTION. SINCE THE EXPRESSION MUST BE STATIC, IT IS REJECTED AT COMPILE TIME.

B86001Y CHECKS FOR A PREDEFINED FIXED-POINT TYPE OTHER THAN DURATION.

C96005B CHECKS FOR VALUES OF TYPE DURATION'BASE THAT ARE OUTSIDE THE RANGE OF DURATION. THERE ARE NO SUCH VALUES FOR THIS IMPLEMENTATION.

CA2009C AND CA2009F CHECK WHETHER A GENERIC UNIT CAN BE INSTANTIATED BEFORE ITS BODY (AND ANY OF ITS SUBUNITS) IS COMPILED; THIS IMPLEMENTATION CREATES A DEPENDENCE ON GENERIC UNITS AS ALLOWED BY AI-00408 AND AI-00506 SUCH THAT THE COMPILATION OF THE GENERIC UNIT BODIES MAKES THE INSTANTIATING UNITS OBSOLETE. (SEE SECTION 2.3.)

CD1009C USES A REPRESENTATION CLAUSE SPECIFYING A NON-DEFAULT SIZE FOR A FLOATING-POINT TYPE.

CD2A84A, CD2A84E, CD2A84I..J (2 TESTS), AND CD2A84O USE REPRESENTATION CLAUSES SPECIFYING NON-DEFAULT SIZES FOR ACCESS TYPES.

BD8001A, BD8003A, BD8004A..B (2 TESTS), AND AD8011A USE MACHINE CODE INSERTIONS.

THE FOLLOWING 264 TESTS CHECK FOR SEQUENTIAL, TEXT, AND DIRECT ACCESS FILES:

| | | | |
|---|---|---|---|
| CE2102A..C (3) | CE2102G..H (2) | CE2102K | CE2102N..Y (12) |
| CE2103C..D (2) | CE2104A..D (4) | CE2105A..B (2) | CE2106A..B (2) |
| CE2107A..H (8) | CE2107L | CE2108A..H (8) | CE2109A..C (3) |
| CE2110A..D (4) | CE2111A..I (9) | CE2115A..B (2) | |
| CE2120A..B (2) | CE2201A..C (3) | EE2201D..E (2) | CE2201F..N (9) |
| CE2203A | CE2204A..D (4) | CE2205A | CE2206A |
| CE2208B | CE2401A..C (3) | EE2401D | CE2401E..F (2) |
| EE2401G | CE2401H..L (5) | CE2403A | CE2404A..B (2) |
| CE2405B | CE2406A | CE2407A..B(2) | CE2408A..B (2) |
| CE2409A..B (2) | CE2410A..B (2) | CE2411A | CE3102A..C (3) |
| CE3102F..H (3) | CE3102J..K (2) | CE3103A | CE3104A..C (3) |
| CE3106A..B (2) | CE3107B | CE3108A..B (2) | CE3109A |
| CE3110A | CE3111A..B (2) | CE3111D..E (2) | CE3112A..D (4) |
| CE3114A..B (2) | CE3115A | CE3119A | |
| EE3203A | EE3204A | CE3207A | CE3208A |
| CE3301A | EE3301B | CE3302A | CE3304A |
| CE3305A | CE3401A | CE3402A | EE3402B |
| CE3402C..D (2) | CE3403A..C (3) | CE3403E..F (2) | CE3404B..D (3) |
| CE3405A | EE3405B | CE3405C..D (2) | CE3406A..D (4) |
| CE3407A..C (3) | CE3408A..C (3) | CE3409A | CE3409C..E (3) |
| EE3409F | CE3410A | CE3410C..E (3) | EE3410F |
| CE3411A | CE3411C | CE3412A | EE3412C |
| CE3413A..C (3) | CE3414A | CE3602A..D (4) | CE3603A |
| CE3604A..B (2) | CE3605A..E (5) | CE3606A..B (2) | |
| CE3704A..F (6) | CE3704M..O (3) | CE3705A..E (5) | CE3706D |
| CE3706F..G (2) | CE3804A..P (16) | CE3805A..B (2) | CE3806A..B (2) |
| CE3806D..E (2) | CE3806G..H (2) | CE3904A..B (2) | CE3905A..C (3) |
| CE3905L | CE3906A..C (3) | CE3906E..F (2) | |

CE2103A..B AND CE3107A EXPECT THAT NAME_ERROR IS RAISED WHEN AN ATTEMPT IS MADE TO CREATE A FILE WITH AN ILLEGAL NAME; THIS IMPLEMENTATION DOES NOT SUPPORT THE CREATION OF EXTERNAL FILES AND SO RAISES USE_ERROR. (See section 2.3.)

## 2.3 TEST MODIFICATIONS

Modifications (see section 1.3) were required for 70 tests.

The following 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.

| | | | | | | |
|---|---|---|---|---|---|---|
| B22003A | B26001A | B26002A | B26005A | B28003A | B29001A | B33301B |
| B35101A | B37106A | B37301B | B37302A | B38003A | B38003B | B38009A |
| B38009B | B55A01A | B61001C | B61001F | B61001H | B61001I | B61001M |
| B61001R | B61001W | B67001H | B83A07A | B83A07B | B83A07C | B83E01C |
| B83E01D | B83E01E | B85001D | B85008D | B91001A | B91002A | B91002B |
| B91002C | B91002D | B91002E | B91002F | B91002G | B91002H | B91002I |
| B91002J | B91002K | B91002L | B95030A | B95061A | B95061F | B95061G |
| B95077A | B97103E | B97104G | BA1001A | BA1101B | BC1109A | BC1109C |
| BC1109D | BC1202A | BC1202F | BC1202G | BE2210A | BE2413A | |

"PRAGMA ELABORATE (REPORT)" has been added at appropriate points in order to solve the elaboration problems for:

C_3030C C86007A

BC3204C and BC3205D were graded passed by Processing Modification as directed by the AVO. These tests check that instantiations of generic units with unconstrained types as generic actual parameters are illegal if the generic bodies contain uses of the types that require a constraint. However, the generic bodies are compiled after the units that contain the instantiations, and this implementation creates a dependence of the instantiating units on the generic units as allowed by AI-00408 and AI-00506 such that the compilation of the generic bodies makes the instantiating units obsolete--no errors are detected. The processing of these tests was modified by re-compiling the obsolete units; all intended errors were then detected by the compiler.

The value used to specify the collection size has been increased from 256 to 580 take alignment into account for:

CD2A83A

CE2103A..B and CE3107A abort with an unhandled exception when USE_ERROR is raised on the attempt to create an external file (see 2.2). The AVO ruled that these tests are to be graded as inapplicable.

# CHAPTER 3

# PROCESSING INFORMATION

## 3.1 TESTING ENVIRONMENT

The executable files were prepared on the VAX host computer chapter by chapter. When a chapter was completely processed, the executables were transferred via an RS232 serial line from the VAX 8530 (host) to the TP860 Single Board Computer (target); this RS232 line is used by a combined downloader/debugger on the host for communication with the monitor on the target. A second RS232 serial line between the host and target is used by the Ada program running on the target board for TEXT_IO communication with a capture program on the host.

A special "run" program is invoked with the name of the absolute file generated by the Ada linker. The "run" program starts two processes: 1) The downloader/debugger loads the absolute program to the board and starts it, using the first RS232 serial line; and, 2) the capture program waits for output from the Ada program on the second RS232 serial line and passes that output on to SYS$OUTPUT on the host.

The downloader/debugger and the on-board monitor are developed by Intel and is called: MED860. The capture program and the "run" program were developed by DDC-I.

The DACS VAX/VMS to 80860 Bare Ada Cross Compiler System, Version 4.6.1 was executed on the target board with the following specfications:

TADPOLE TECHNOLOGY PLC TP860M (Bare Board)
One internal timer
Two serial ports
4MB RAM

For each chapter, a command file was generated that loaded and executed every program.

For a point of contact for technical information about this Ada implementation system, see:

In the U.S.A.:

Mr. Tom Gender
DDC-I, Inc.
9630 North 25th Avenue

Suite #118
Phoenix, Arizona 85021

Mailing address:
P.O. Box 37767
Phoenix, Arizona 85069-7767
Telephone: 602-944-1883
Telefax: 602-944-3253

In the rest of the world:

Mr. Ole Dommergaard
DDC International A/S
Gl. Lundtoftevej 1B
DK-2800 Lyngby
DENMARK

Telephone: + 45 45 87 11 44
Telefax: + 45 45 87 22 17

For a point of contact for sales information about this Ada implementation system, see:

In the U.S.A.:

Mr. Mike Turner
DDC-I, Inc.
9630 North 25th Avenue
Suite #118
Phoenix, Arizona 85021

Mailing address:
P.O. Box 37767
Phoenix, Arizona 85069-7767
Telephone: 602-944-1883
Telefax: 602-944-3253

In the rest of the world:

Mr. Palle Andersson
DDC International A/S
Gl. Lundtoftevej 1B
DK-2800 LYNGBY
Denmark

Telephone: + 45 45 87 11 44
Telefax: + 45 45 87 22 17

Testing of this Ada implementation was conducted at the

customer's site by a validation team from the AVF.

## 3.2 SUMMARY OF TEST RESULTS

An Ada Implementation passes a given ACVC version if it processes each test of the customized test suite in accordance with the Ada Programming Language Standard, whether the test is applicable or inapplicable; otherwise, the Ada Implementation fails the ACVC [Pro90].

For all processed tests (inapplicable and applicable), a result was obtained that conforms to the Ada Programming Language Standard.

| | | | |
|---|---|---|---|
| a) | Total Number of Applicable Tests | 3549 | |
| b) | Total Number of Withdrawn Tests | 93 | |
| c) | Processed Inapplicable Tests | 528 | |
| d) | Non-Processed I/O Tests | 0 | |
| e) | Non-Processed Floating-Point Precision Tests | 0 | |
| f) | Total Number of Inapplicable Tests | 528 | (c+d+e) |
| g) | Total Number of Tests for ACVC 1.11 | 4170 | (a+b+f) |

## 3.3 TEST EXECUTION

Version 1.11 of the ACVC comprises 4170 tests. When this compiler was tested, the tests listed in section 2.1 had been withdrawn because of test errors. The AVF determined that 528 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing. In addition, the modified tests mentioned in section 2.3 were also processed.

A magnetic tape containing the customized test suite (see section 1.3) was taken on-site by the validation team for processing. The tests were compiled and linked on the host computer system, as appropriate. The executable images were transferred to the target computer system by the communications link described above, and run. The results were captured on the host computer system using the communications link described above.

Testing was performed using command scripts provided by the customer and reviewed by the validation team. See Appendix B for a complete listing of the processing options for this implementation. It also indicates the default options. The options invoked explicitly for validation testing during this test were:

```
/LIST   /NOSAVE_SOURCE
```

The options invoked by default for validation testing during this test were:

```
/CHECK   /CONFIGURATION_FILE = <default file>
/NOTARGET_DEBUG   /LIBRARY    /NOOPTOMIZE
/NOPROGRESS   /NOXREF
```

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

# APPENDIX A

## MACRO PARAMETERS

This appendix contains the macro parameters used for customizing the ACVC. The meaning and purpose of these parameters are explained in [UG89]. 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 132 the value for $MAX_IN_LEN--also listed here. These values are expressed here as Ada string aggregates, where "V" represents the maximum input-line length.

| Macro Parameter | Macro Value |
|---|---|
| $MAX_IN_LEN | 132 |
| $BIG_ID1 | (1..V-1 => 'A', V => '1') |
| $BIG_ID2 | (1..V-1 => 'A', V => '2') |
| $BIG_ID3 | (1..V/2 => 'A') & '3' & (1..V-1-V/2 => 'A') |
| $BIG_ID4 | (1..V/2 => 'A') & '4' & (1..V-1-V/2 => 'A') |
| $BIG_INT_LIT | (1..V-3 => '0') & "298" |
| $BIG_REAL_LIT | (1..V-5 => '0') & "690.0" |
| $BIG_STRING1 | '"' & (1..V/2 => 'A') & '"' |
| $BIG_STRING2 | '"' & (1..V-1-V/2 => 'A') & '1' & '"' |
| $BLANKS | (1..V-20 => ' ') |
| $MAX_LEN_INT_BASED_LITERAL | "2:" & (1..V-5 => '0') & "11:" |
| $MAX_LEN_REAL_BASED_LITERAL | "16:" & (1..V-7 => '0') & "F.E:" |
| $MAX_STRING_LITERAL | '"' & (1..V-2 => 'A') & '"' |

The following table contains the values for the remaining macro parameters.

| Macro Parameter | Macro Value |
|---|---|

```
ACC_SIZE                                : 32
ALIGNMENT                               : 4
COUNT_LAST                              : 2_147_483_647
DEFAULT_MEM_SIZE                        : 2#1#E32
DEFAULT_STOR_UNIT                       : 8
DEFAULT_SYS_NAME                        : DACS_80860
DELTA_DOC                               : 2#1.0#E-31
ENTRY_ADDRESS                           : system."-"(16#7fcafc00#)
ENTRY_ADDRESS1                          : system."-"(16#7fcafb00#)
ENTRY_ADDRESS2                          : system."-"(16#7fcafa00#)
FIELD_LAST                              : 67
FILE_TERMINATOR                         : ' '
FIXED_NAME                              : NO_SUCH_TYPE
FLOAT_NAME                              : NO_SUCH_TYPE
FORM_STRING                             : ""
FORM_STRING2                            : "EXTERNAL FILES NOT SUPPORTED"
GREATER_THAN_DURATION                   : 100_000.0
GREATER_THAN_DURATION_BASE_LAST         : 200_000.0
GREATER_THAN_FLOAT_BASE_LAST            : 1.80141E+38
GREATER_THAN_FLOAT_SAFE_LARGE           : 1.0E308
GREATER_THAN_SHORT_FLOAT_SAFE_LARGE     : 1.0E308
HIGH_PRIORITY                           : 31
ILLEGAL_EXTERNAL_FILE_NAME1             : #1\NODIRECTORY\FILENAME
ILLEGAL_EXTERNAL_FILE_NAME2             :
   THIS_FILE_NAME_FAR_IS_TOO_LONG_FOR_THIS_IMLPEMENTATION
INAPPROPRIATE_LINE_LENGTH               : -1
INAPPROPRIATE_PAGE_LENGTH               : -1
INCLUDE_PRAGMA1                         :
   PRAGMA INCLUDE ("A28006D1.TST")
INCLUDE_PRAGMA2                         :
   PRAGMA INCLUDE ("B28006E1.TST")
INTEGER_FIRST                           : -2147483648
INTEGER_LAST                            :  2147483647
INTEGER_LAST_PLUS_1                     : 2_147_483_648
INTERFACE_LANGUAGE                      : AS860
LESS_THAN_DURATION                      : -100_000.0
LESS_THAN_DURATION_BASE_FIRST           : -200_000.0
LINE_TERMINATOR                         : ' '
LOW_PRIORITY                            : 0
MACHINE_CODE_STATEMENT                  : NULL;
MACHINE_CODE_TYPE                       : NO_SUCH_TYPE
MANTISSA_DOC                            : 31
MAX_DIGITS                              : 15
MAX_INT                                 : 2147483647
```

```
MAX_INT_PLUS_1                  : 2_147_483_648
MIN_INT                         : -2147483648
NAME                            : NO_SUCH_TYPE_AVAILABLE
NAME_LIST                       : DACS_80860
NAME_SPECIFICATION1             : EXTERNAL_FILES_NOT_SUPPORTED
NAME_SPECIFICATION2             : EXTERNAL_FILES_NOT_SUPPORTED
NAME_SPECIFICATION3             : EXTERNAL_FILES_NOT_SUPPORTED
NEG_BASED_INT                   : 16#F000000E#
NEW_MEM_SIZE                    : 2#1#E32
NEW_STOR_UNIT                   : 8
NEW_SYS_NAME                    : DACS_80860
PAGE_TERMINATOR                 : ' '
RECORD_DEFINITION               : NEW INTEGER;
RECORD_NAME                     : NO_SUCH_MACHINE_CODE_TYPE
TASK_SIZE                       : 32
TASK_STORAGE_SIZE               : 1024
TICK                            : 0.000_001
VARIABLE_ADDRESS                : system."-"(16#fcaff00#)
VARIABLE_ADDRESS1               : system."-"(16#fcafe00#)
VARIABLE_ADDRESS2               : system."-"(16#fcafd00#)
YOUR_PRAGMA                     : INTERFACE_SPELLING
```

# APPENDIX B

## 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.

| QUALIFIER | DESCRIPTION |
|---|---|
| /AUTO_INLINE<br>/NOAUTO_INLINE | Small local subprograms are automatically inline expanded. |
| /CHECK<br>/NOCHECK | Generates run-time constraint checks. |
| /CONFIGURATION_FILE | Specifies the file used by the compiler. |
| /EXCEPTION_TABLES<br>/NOEXCEPTION_TABLES | Controls the generation of exception name tables. |
| /LIBRARY | Specifies program library used. |
| /LIST<br>/NOLIST | Creates a source list file. |
| /MACHINE_CODE<br>/NOMACHINE_CODE | Displays a machine code listing. |
| /OPTIMIZE<br>/NOOPTIMIZE | Specifies compiler optimizations. |
| /PROGRESS<br>/NOPROGRESS | Displays compiler progress. |
| /SAVE_SOURCE<br>/NOSAVE_SOURCE | Copies source to program library. |
| /SORT_OBJECTS<br>/NOSORT_OBJECTS | Specifies that objects are allocated according to their size. |
| /SYNTAX | Specifies that only syntax checking should be performed. |
| /TRACEBACK<br>/NOTRACEBACK | Controls whether or not symbolic traceback tables are generated. |

/UNIT — Assigns a specific unit number to the compilation (must be free and in a sublibrary).

/XREF — Creates a cross reference listing.
/NOXREF

## 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.

The link process is controlled by a variety of parameters, qualifiers, and logical names. The following diagram illustrates the linking process:

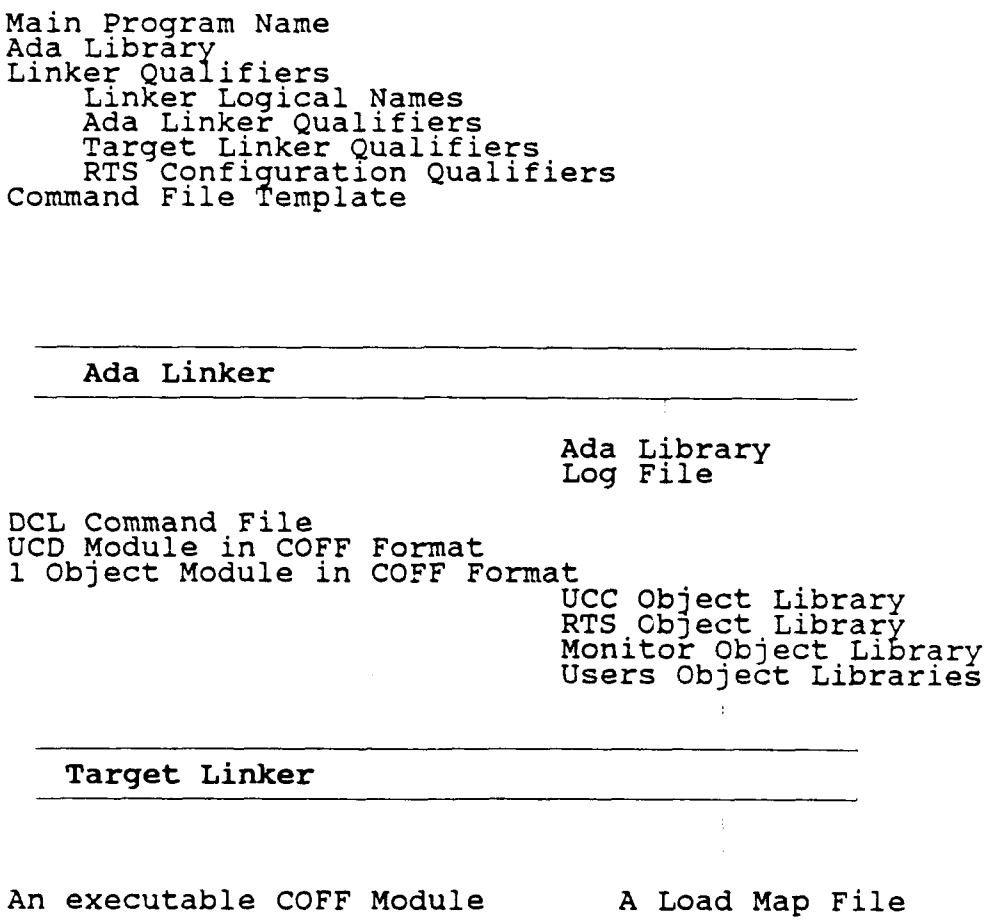


Figure 6-1
The Linking Process

## A. Invocation Command

Enter the following command at DCL level to invoke the linker:

```
$ ADA/LINK {<qualifier>}  <unit-name>
```

The qualifiers, parameters, and logical names supported by the linker are described in the following sections.

### 1. Parameter

<unit-name>

This parameter is required and indicates the main program. The <unit_name> must be a library unit in the current program library, but not necessarily of the current sublibrary.

Note, that a main program must be a procedure without parameters, and that <unit-name> is the identifier of the procedure, not a file specification. The main procedure is not checked for parameters, but the execution of a program with a main procedure with parameters is undefined.

### 2. Logical Names Used by the Ada Linker

When a link is executed, the following logical names are used:

| LOGICAL NAME | PURPOSE |
|---|---|
| ADA_SYSTEM_LIBRARY | Identifies the root library where the system compilation units reside. |
| ADA_LIBRARY | Identifies the default library used by all DACS-80860 tools. It is the lowest level sublibrary in the program library hierarchy. |
| ADA_TEMPLATE | Identifies the template file used to control the target part of the link. |
| ADA_LINK_DEFAULT | Specifies the other default values for the linker. |

| | |
|---|---|
| ADA_RTS_LIB | Identifies the RTS object library for the Permanent Part of the Run-Time System. |
| ADA_UCC_LIB | Identifies the RTS object library for the User Configurable Code portion of the Run-Time System. |
| ADA_MON_LIB | Identifies the object library containing the interface to the onboard monitor. |

With each of these logical names, the part of the name in boldface will differ depending on the command verb defined at installation. This example uses "ADA".

## 3. Summary of Qualifiers

This section briefly describes all qualifiers supported by the Ada linker. They are placed into three categories, those that affect the:

1) Ada link phase
2) Target link phase
3) Configuration of the RTS

All qualifier names may be abbreviated (characters omitted from the right) as long as no ambiguity arises (standard VAX/VMS DCL convention).

### a. Ada Link Qualifiers

These options affect only the Ada part of the link process.

| OPTION | DESCRIPTION |
|---|---|
| **/EXCEPTION_TABLES**<br>**/NOEXCEPTION_TABLES** | Specifies whether or not tables containing the spelling of user defined exception should be included in the executable file. |
| **/LIBRARY** | The library used in the link |

| | |
|---|---|
| /LOG<br>/NOLOG | Specifies creation of a log file |
| /STOP_BEFORE_TARGET_LINK<br>/NOSTOP_BEFORE_TARGET_LINK | Performs Ada link only |
| /TEMPLATE | Specifies template file for the target link |
| /WARNINGS<br>/NOWARNINGS | Specifies whether or not warnings are displayed |

b. Target Link Qualifiers

These options control the target part of the link process.

| OPTION | DESCRIPTION |
|---|---|
| /BSSADDR | Specfies the RAM address to be used for common blocks and for the bss section that follows |
| /DATAADDR | Specifies the address at which the data section is to be loaded |
| /ENTRYSYMBOL | Specifies a symbol as the entry point |
| /I386 | Inserts the magic number for the Intel386 architecture in the output object file, so that 80386-oriented tools can process the executable module |
| /INCREMENTAL | Retains the relocation entries in the output file, which enables incremental linking |
| /INITSYM | Initializes the symbol table with the specified symbol |

| | |
|---|---|
| /LENDATA | Specifies the length of the data section as integer bytes |
| /LISTFILE | Specifies a list of object files which should be included in the linking process |
| /MAP | Generates a load map |
| /MONITOR_LIBRARY | Specifies the object library containing the interface modules to the onboard monitor. |
| /NONPAGEABLE | Starts the text and data sections exactly at the address specified by the TEXTADDR and DATAADDR options without performing the normal modifications to those addresses to make the file pageable |
| /OUTFILE | Specifies the name of the output module |
| /PAGEALIGN | Aligns the data sections on a page boundary |
| /RTS_LIBRARY | Specifies the object library containing the runtime system. |
| /STRIPSYM | Strips all symbols from the output object file |
| /TARGET_OPTIONS | Specifies a string which is passed to the template without interpretation. |
| /TEXTADDR | Specifies the address at which the text section is to be loaded |
| /UCC_LIBRARY | Specifies the object library containing the UCC modules. |
| /USR_LIBRARY | Target libraries or object modules to include in target link |
| /XMAP | Generates a load map including cross-references |

c. Run-Time System Qualifiers

These qualifiers are used to configure the RTS and are described in more detail in chapter 7.

| OPTION | DESCRIPTION |
|---|---|
| /HEAP_ADDRESSES | Specifies addresses of heap areas. |
| /HEAP_SIZE | Specifies size of heap areas. |
| /INT_STACK_SIZE<br>/NOINT_STACK_SIZE | Amount of stack space reserved to handle user interrupts |
| /INT_TASKS<br>/NOINT_TASKS | Specifies the number of interrupt tasks in the application |
| /MP_STACK_SIZE | Main program stack size |
| /PRIORITY | Default task priority |
| /TASK_STACK_SIZE | Default stack size for all tasks |
| /TASKS<br>/NOTASKS | Maximum number of tasks or non-tasking application |
| /TIME_SLICE<br>/NOTIME_SLICE | Task time slicing enabled/disabled and optional time slice value |
| /TIMER | Timer resolution |
| /TRACEBACK<br>/NOTRACEBACK | Enables traceback when a program has an unhandled exception |
| /UCD<br>/NOUCD | Controls whether or not a User Configurable Data module is generated |

# APPENDIX C

## 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 the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics 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. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

```
package STANDARD is

type SHORT_INTEGER is range -32_768 .. 32_767;

type INTEGER is range -2_147_483_648 .. 2_147_483_647;

type FLOAT is digits 6
     range -16#0.FFFF_FF#E32 .. 16#0.FFFF_FF#E32;

type LONG_FLOAT is digits 15
     range -16#0.FFFF_FFFF_FFFF_F8#E256 ..
           16#0.FFFF_FFFF_FFFF_F8#E256;

type DURATION is delta 2#1.0#E-14 range -131_072.0 .. 131_071.0;

end STANDARD;
```

# APPENDIX I. - IMPLEMENTATION DEPENDENT CHARACTERISTICS

This appendix describes the implementation-dependent characteristics of DACS-80860 as required in Appendix F of the Ada Reference Manual (ANSI/MIL-STD-1815A).

## A. Implementation-Dependent Pragmas

This section describes all implementation defined pragmas.

### 1. Pragma INTERFACE SPELLING

This pragma allows an Ada program to call a non-Ada program whose name contains characters that would be an invalid Ada subprogram identifier. This pragma must be used in conjunction with pragma INTERFACE, i.e., pragma INTERFACE must be specified for the non-Ada subprogram name prior to using pragma INTERFACE_SPELLING.

#### a. Format

The pragma has the format:

```
pragma INTERFACE_SPELLING (subprogram name, string literal);
```

where the subprogram name is that of one previously given in pragma INTERFACE and the string literal is the exact spelling of the interfaced subprogram in its native language. This pragma is only required when the subprogram name contains invalid characters for Ada identifiers.

#### b. Example

```
function ASSEMBLY_MODULE_NAME return INTEGER;

pragma INTERFACE (AS860, ASSEMBLY_MODULE_NAME);
pragma INTERFACE_SPELLING (ASSEMBLY_MODULE_NAME,
                           "Illegal$Ada_Name");
```

## B. Implementation-Dependent Attributes

No implementation-dependent attributes are defined.

## C. Package SYSTEM

The package SYSTEM is described in ARM 13.4.

```
package  SYSTEM  is

   type ADDRESS is new INTEGER;
   type NAME is (DACS_80860);

   SYSTEM_NAME      : constant NAME  := DACS_80860;
   STORAGE_UNIT     : constant       := 8;
   MEMORY_SIZE      : constant       := 2#1#E32;
   MIN_INT          : constant       := -2_147_483_648;
   MAX_INT          : constant       := 2_147_483_647;
   MAX_DIGITS       : constant       := 15
   MAX_MANTISSA     : constant       := 31
   FINE_DELTA       : constant       := 2#1.0#E-31;
   TICK             : constant       := 0.000_001;

   subtype PRIORITY is INTEGER range 0..31;
   type INTERFACE_LANGUAGE is (AS860);  -- implementation dependent

end SYSTEM;
```

## D. Representation Clauses

The representation clauses that are accepted are described below. Note that representation specifications can be given on derived types too.

The DDC-I Ada Compiler System conforms to the defined standard for language conventions of the i860 microprocessor ensuring that application program written by different people and organizations will work together. See [Intel PRM].

### 1. Length Clause

Four kinds of length clauses are accepted.

**Size specifications:**

The size attribute for a type T is accepted in the following cases:

- If T is a discrete type then the specified size must be greater than or equal to the number of bits needed to represent a value of the type, and less than or equal to 32. Note that when the number of bits needed to hold any value of the type is calculated, the range is extended to include 0 if necessary, i.e. the range 3..4 **cannot** be represented in 1 bit, but needs 3 bits.

- If T is a fixed point type, then the specified size must be greater than or equal to the smallest number of bits needed to hold any value of the fixed point type, and less than 32 bits. Note that the Reference Manual permits a representation, where the lower bound and the upper bound is not representable in the type. Thus the type

  ```
  type FIX is delta 1.0 range -1.0 .. 7.0;
  ```

  is representable in 3 bits. As for discrete types, the number of bits needed for a fixed point type is calculated using the range of the fixed point type possibly extended to include 0.0.

- If T is a floating point type, an access type or a task type the specified size must be equal to the number of bits used to represent values of the type per default (floating points: 32 or 64, access types : 32 bits and task types : 32 bits).

- If T is a record type the specified size must be greater or equal to the minimal number of bits used to represent values of the type per default.

- If T is an array type the size of the array must be static i.e. known at compile time and the specified size must b equal to the minimal number of bits used to represent values of the type per default.

The size given in the length clause will be used when allocating space for values of the type in all contexts e.g. as part of an array or record. For declared objects the size will be rounded to the nearest number of bytes before the object is allocated.

**Collection size specifications:**

Using the STORAGE_SIZE attribute on an access type will set an upper limit on the total size of objects allocated int he collection allocated for the access type. If further allocation is attempted, the exception STORAGE_ERROR is raised. The specified storage size must be less than or equal to INTEGER'LAST

**Task storage size**

When the STORAGE_SIZE attribute is given on a task type, the task stack area will be of the specified size. The specified storage size must be less than or equal to INTEGER'LAST.

**Small specifications**

Any value of the SMALL attribute less than the specified delta for the fixed point type can be given.

## 2. Enumeration Representation Clauses

Enumeration representation clauses may specify representations in the range of SHORT_INTEGER'FIRST .. SHORT_INTEGER'LAST. An enumeration representation clause may be combined with a length clause. If an enumeration representation clause has been given for a type the representational values are considered when the number of bits needed to hold any value of the type is evaluated. Thus the type

```
type ENUM is (A,B,C);
for ENUM use (1,2,3);
```

needs 3 bits to represent any value of the type.

## 3. Record Representation Clauses

When component clauses are applied to a record type, the following should be noted:

- Components can start at any bit boundary. Placing e.g. non packed arrays on odd bit boundaries will cause costly implicit conversion to be generated, however.

- All values of the component type must be representable within the specified number of bits in the component clause.

- If the component type is either a discrete type or a fixed point type, then the component is packed into the specified number of bits (see however the restriction in the paragraph above).

- If the component type is not one of the types specified in the paragraph above, the default size calculated by the compiler must be given as the bit width, i.e. the component must be specified as

  ```
  component at N range X..X + component_type'SIZE - 1
  ```

  where N specifies the relative storage unit number (0,1,...) from the beginning of the record, and X is any bit number.

- The maximum bit width for components of scalar types is 32.

- The maximum bit width for all component types is 32 bits.

If the record type contains components which are not covered by a component clause, they are allocated consecutively after the component with the highest offset specified by a component clause. Holes created because of component clauses are not otherwise utilized by the compiler.

When the compiler determines the size of a record component the following is taken into account in the specified order:

. a component clause
. a length clause ('SIZE) on the component type
. a possible pragma PACK on the record type
. the default size of the component type

### a. Alignment Clauses

Alignment clauses for records are supported with the following restrictions:

- The specified alignment boundary must be 1,2,4,8 or 16.

- The specified alignment must not conflict with the alignment requirement for the record components, i.e. an alignment boundary of 4 is not accepted if the record has a component of an array type with size 100 bytes (such arrays should be aligned on a 16 byte boundary).

## E. Names for Implementation-Dependent Components

None defined by the compiler.

## F. Address Clauses

Address clauses are supported for scalar and for composite objects whose size can be determined at compile time.

## G. Unchecked Conversion

Unchecked conversion is only allowed between objects of the same "size". However, if scalar type has different sizes (packed and

unpacked), unchecked conversion between such a type and another type is accepted if either the packed or the unpacked size fits the other type.

## H. Input/Output Packages

In many embedded systems, there is no need for a traditional I/O system, but in order to support testing and validation, DDC-I has developed a small terminal oriented I/O system. This I/O system consists essentially of TEXT_IO adapted with respect to handling only a terminal and not file I/O (file I/O will cause a USE error to be raised) and a low level package called TERMINAL_DRIVER. A BASIC_IO package has been provided for convenience purposes, forming an interface between TEXT_IO and TERMINAL_DRIVER as illustrated in the following figure.