

# *IEEE Standards Interpretations*

## IEEE Std 1076-1987 IEEE Standard VHDL Language Reference Manual

Sponsor

Design Automation Technical Committee  
of the  
IEEE Computer Society

**Abstract:** A series of analyses of various problems regarding the interpretation of the VHSIC Hardware Description Language (VHDL) was carried out by VHDL experts on the VHDL Analysis and Standardization Group, the working group that developed the original IEEE Std 1076-1987. The results of these deliberations are presented in this document. The intent is to give the VHDL community reasonable ways of interpreting the unclear portions of the standard.

**Keywords:** interpretations, LRM, VHDL, VHSIC Hardware Description Language

The Institute of Electrical and Electronics Engineers, Inc.  
345 East 47th Street, New York, NY 10017-2394, USA

Copyright © 1992 by the  
Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved. Published 1992  
Printed in the United States of America

ISBN 1-55937-181-1

*No part of this publication may be reproduced in any form,  
in an electronic retrieval system or otherwise,  
without the prior written permission of the publisher.*

## Library of Congress Cataloging-in-Publication Data

Institute of Electrical and Electronics Engineers.

IEEE standards interpretations : IEEE Std 1076-1987, IEEE standard  
VHDL language reference manual / sponsor, Design Automation  
Technical Committee of the IEEE Computer Society.

p. cm.  
"1076/INT-1991"

Includes index.

ISBN 1-55937-181-1

1. VHDL (Computer hardware description language) I. IEEE  
Computer Society. Design Automation Technical Committee.

II. Title.

TK7885.7.I58 1991  
621.39'2—dc20

91-40348  
CIP

## IEEE Standards Interpretations

Occasionally, questions may arise regarding the meaning of portions of standards as they relate to specific applications. Such requests for interpretations should ask for clarifications of the exact nature of the contents of the standard.

Questions relating to such interpretations are reviewed and evaluated by a balance of members representing the specific committee interests. This officially formed interpretations subgroup creates and circulates a draft interpretation among its members. It transmits a final interpretation to the party initiating the request only after consensus has been achieved. The interpretation is also given to the standards-developing committee to consider its inclusion in a supplement or the next revision to the standard.

Interpretations are issued to explain and clarify the intent of the standard and are not intended to constitute an alteration to the original standard or to supply consulting information. The interpretations subgroup cannot make new rules to fit situations not yet covered in the standard, even if the investigations of the subgroup lead it to conclude that the requirement is incomplete or in error. Changes to the standard are made only through revisions or supplements to the standard.

It is recognized that requests are frequently received that are partially or totally requests for information rather than requests for an interpretation. It is inappropriate to issue an official interpretation to answer such requests. The interpretations subgroup may, however, find from its research that the literal printing of the standards text is not identical to that approved by the standards developers and may issue an editorial correction as a part of its interpretation.

The original interpretations requests in this document have been lightly edited to remove extraneous matter and to focus on the problem presented. Some illustrations have been redrawn for publication. With these exceptions, requests are in the form received.

## Procedure for Requesting an Interpretation

Requests for interpretations should be addressed to

Secretary, IEEE Standards Board  
IEEE Standards Department  
445 Hoes Lane  
P.O. Box 1331  
Piscataway, NJ 08855-1331

Requests for interpretations should include

- (1) The specific designation of the standard, including the year of publication.
- (2) The specific subsection being questioned.
- (3) The applicable conditions for the case in question.

Line drawings should be black ink or excellent black pencil originals. Photos should be black-and-white glossy prints. These illustrations must be reproduced for committee circulation and eventually will be used to supplement the text of the next edition. Clear diagrams and pictures will make the work of interpretation easier and more valuable to users.

Requests, including all supplementary material, must be in a form that is easily reproducible. If suitable for consideration, requests will be sent to the interpretations subgroup. After consideration by the subcommittee, which may involve many exchanges of correspondence, the inquirer will be notified of the decision of the subgroup. Decisions will be published from time to time in cumulative form and may be ordered from the IEEE.

## Foreword

In 1987 the IEEE standardized the VHSIC Hardware Description Language (VHDL). This standardization was a remarkable effort, as a very complex language standard was developed and balloted within two years. This is a tribute both to the many volunteers who participated in the efforts and to the work done by CLSI under contract to the Air Force.

In addition, the results of this standardization process, IEEE Std 1076-1987 (the language that has come to be known as VHDL 1076), has achieved wide acceptance throughout the industry. Scarcely a month goes by without the announcement of several new VHDL tools.

All of this is encouraging to those who believe that an industry-wide standard way of describing hardware is essential. However, the speed of the initial standardization of VHDL and its wide acceptance also raises a new problem: interpretation of the standard. Two years is too short a time to iron out all the ambiguities of any useful HDL, not to mention one as complex as VHDL. Thus, it is not surprising that important parts of IEEE Std 1076-1987 have been given differing interpretations. However, as understandable as this situation is, it is clearly unacceptable. A standard with multiple differing implementations is almost as bad as no standard, and, indeed, could threaten the continued acceptance of VHDL in the industry.

The “obvious” solution to this problem would be to amend IEEE Std 1076-1987, the VHDL Language Reference Manual (LRM), to fix the deficient sections. Unfortunately, this is not possible in practice, since the rules of the IEEE are such that a standard can be changed only by reballoting it—a major effort. Since adding “patches” to the IEEE 1076 VHDL LRM would amount to changing the standard, that course was effectively eliminated in the short term.

The solution to this problem was to develop a series of analyses of the various problems by VHDL experts in the VHDL Analysis and Standardization Group (VASG)—the working group that had developed IEEE Std 1076-1987. The decisions of this group could not actually alter the standard, but would give the VHDL community a means of interpreting the unclear portions of the standard. Moreover, given that the VASG is made up of a broad range of expert representatives of the major VHDL vendors and users, the consensus reached by this group are standard interpretations of IEEE Std 1076-1987, the VHDL LRM.

In practice, a subgroup of the VASG, known as the Issues Screening and Analysis Committee (ISAC), was formed as soon as IEEE Std 1076-1987 was approved. This committee has had a varying membership over the years. Despite this, it has diligently reviewed the sometimes esoteric issue reports (IRs) generated by VHDL users and tool developers. The practice of the ISAC has been first to seek a consensus within itself for every suggested interpretation, and only then report to the VASG as a whole. Only those ISAC interpretations that have gained a clear consensus from the whole of the VASG have been adopted. Indeed, there has been at least one instance where only one objection was raised to a suggested solution, but that minority vote caused a reformulated position to be developed.

The current document is the result of the ISAC and VASG deliberations. It is an official interpretation but not, as previously stated, an official amendment to IEEE Std 1076-1987. Rather, it represents the best analysis of experts in the area. One does not contravene the VHDL standard if one chooses to ignore the interpretations herein. However, doing so both contradicts the spirit of cross-industrial cooperation in which VHDL was born and has thrived and counters the direction of the major VHDL vendors, all of whom have active representatives on the ISAC and the VASG.

## Introduction

Three sorts of errors were discernable in the following IRs. First, there were straightforward errors—usually mere typographical errors. These were easily dispatched. Second, there were places where the standard is vague on a given point. In some cases, these could be resolved by consulting with the original designers of VHDL to see what their intention had been. If such consultation did not explain the matter or raised other issues, then further substantial analysis was required.

Finally, there are the cases where the standard contradicts itself. A subtle variation on this contradiction theme is to be found in the cases where rules are established for a general language construct, while a subcase of the general construct has conflicting rules given to it. Much consideration has been required to resolve this sort of issue.

Although the IRs have serial numbers that are without gaps, the ISAC did not address the issues in serial-number order. Instead, an effort was made to resolve important ones first, with some attention also given to those that were “easily” resolved. Consequently, although the IRs contained herein are presented in serial-number order, there are gaps in the sequence. In every case, these gaps represent IRs that have not been resolved by the VASG. Copies of these unresolved IRs can be obtained by contacting the VASG Chair.

Note that each IR in this document has a section indicating a “suggested changes to the language” for the next version of the standard. Several points need to be made regarding this section. First, it reflects the inability of the VASG to change the standard without a taking new ballot. On more than a few occasions, the preferred solution to a language problem was the alteration of the language. Lacking the ability to make such changes, the ISAC opted for a resolution that stayed within the bounds of the current language, but still noted the preferred solution.

Second, it should be noted that no inference should be made that any of these “suggested future changes” will actually appear in a future version of the standard. It may turn out that what seemed to be a good change could not be readily implemented or perhaps conflicts with other desired changes. Indeed, the suggestions for future language revisions were never voted upon, although a number of them were discussed during VASG meetings. The reader should, therefore, take the “VASG-ISAC Interpretation of IEEE Std 1076-1987” section of each IR as the complete and official interpretation of VHDL 1076; the “VASG-ISAC Recommendation for Future Revisions” should be examined for informational purposes only.

The members of the Issues Screening and Analysis Group were:

### **Doug Dunlop, Chair**

Mark Alexandre  
David Bernstein  
Clive Charlwood  
Doug Dunlop  
Larry Groves

Ray Hookway  
Oz Levia  
Bob Mackey  
Paul Menchini  
Ken Scott

Alec Stanculescu  
Charles Swart  
Andy Tsay  
Cary Ussery  
Alex Zamfirescu

The members of the VHDL Analysis and Standardization Group that approved this document were:

**Stan Krolikoski, *Chair***

Mart Altmae  
Jim Armstrong  
John Aynsley  
Steve Bailey  
Peter Barck  
Dave Barton  
Jean-Michel Bergé  
Victor Berman  
Raymond Bouté  
Mark Brown  
Clive Charlwood  
Dave Coelho  
Ted Corman  
Robert Cottrell  
Antonie De Jager  
Joanne DeGroat  
Doug Dunlop  
Hazem El Tahawi  
Jacques Flandrois  
Al Gilman  
Steve Grout

Andrew Guyler  
Paul Harper  
Randy Harr  
John Hillawi  
John Hines  
Andreas Hohl  
Ray Hookway  
Hilary Kahn  
Jean François Lambert  
Kin Sing Lau  
Claude Le Faou  
Oz Levia  
Al Lowenstein  
Erich Marschner  
Paul Menchini  
Jean Mermet  
Wolfgang Nebel  
Kevin Nolan  
Andreas Oszko  
Adam Pawlak  
Robert Piloty

Adam Postula  
Pierre Ragon  
Franz Rammig  
Debbie Rooney  
Jacques Rouillard  
Ashral Salem  
Larry Saunders  
Jean-Pierre Schoellkopf  
Ken Scott  
David Sellars  
Moe Shahdad  
Alec Stanculescu  
Chuck Swart  
Jacques Tete  
Andy Tsay  
Cary Ussery  
Eugenio Villar  
Karen Watkins  
Ron Waxman  
John Winkler  
Alex Zamfirescu

Mary Lynne Nielsen  
*IEEE Standards Department Project Editor*

## A Guide to the IEEE Std 1076-1987 Interpretations Issue Reports

Issues are reported in a standard, machine-readable format, called the “issue report.” The issue reports presented herein are derived from this machine-readable source, as modified by minor editing and formatting. Each issue report begins with a header section, which supplies basic information about the issue. The remainder of the issue report consists of the sections “Description of the Problem,” “Proposed Resolution,” “VASG-ISAC Analysis and Rationale,” “VASG-ISAC Interpretation for IEEE Std 1076-1987,” and “VASG-ISAC Recommendation for Future Revisions.” Each is explained below.

### Issue Report Header Section

The issue report header contains the following fields:

**Classification:** The information contained in this entry consists of one of the following:

*Examples, Notes, and Appendixes* The issue author believes that there is an error in one or more LRM examples, notes or appendixes. As these sections do not define VHDL in any way, but merely serve to illustrate or expand on a point (i.e., they are “nonnormative”), these are minor errors.

*Language Deficiencies and Modeling Problems* The issue author maintains that VHDL is insufficiently expressive or cannot easily or conveniently model some feature or system of interest. These issues were collected, but due to the charter of the ISAC, they were not typically resolved by the ISAC. Instead, they were passed on to the VASG as a whole for resolution during the revision of IEEE Std 1076-1987.

*Language Definition* Also known as “Language Clarification.” The issue author believes that the definition of VHDL, as expressed in IEEE Std 1076-1987, is ambiguous, contradictory, underspecified, or otherwise ill-formed.

*Terminology, Grammar, and Typographical Errors* Also known as “Typographical and Editorial Errors,” “Typographical Error,” “Note Error,” or just “Editorial.” The issue author reports that the standard contains one or more misspellings, grammatical errors, or formatting mistakes affecting normative sections. These are not serious enough to hide the intent of the passage in question.

This field is typically filled in by the author, but may be modified by the ISAC during the course of their deliberations.

**Summary:** This field contains a short (usually one-sentence) description of the issue. This field is typically filled in by the author, but may be modified by the ISAC during the course of their deliberations.

**Related IRs:** Occasionally, the same issue will be reported several times. More often, multiple issues will be related in some way by a language construct or another, similar language feature. This field serves as a cross-reference to another issue report having some bearing on this one. This field is

typically filled in by the ISAC, although issue report authors are encouraged to fill it in if they wish.

**Relevant Sections:** The sections of IEEE Std 1076-1987 affected or cited by this issue report are supplied here. This field is typically filled in by issue authors and augmented by the ISAC during their deliberations.

**Key Words/Phrases:** Any key words or phrases denoting LRM or VHDL terminology are filled in in this field. This field is used to cross-reference issues that affect the same parts of VHDL. This field is typically filled in by issue authors and augmented by the ISAC during their deliberations.

**Current Status:** This field, maintained by the ISAC, contains one of the following values:

*Submitted* The issue author has assembled the issue report and submitted it to the VASG, and it has been installed in the ISAC data base.

*Analyzed* The ISAC has discussed the issue report and has recorded its thoughts in the “VASG-ISAC Analysis and Rationale” section. This status indicates that the ISAC has not reached a consensus on the proper resolution of the issue.

*ISAC-Approved* The ISAC has discussed the issue report, recorded its thoughts in the “VASG-ISAC Analysis and Rationale” section, and recorded its recommended resolution for IEEE Std 1076-1987 in the “VASG-ISAC Interpretation for IEEE Std 1076-1987” section; the ISAC may also have recorded other observations, comments, and recommendations in the “VASG-ISAC Recommendation for Future Revisions” section. The issue report has been passed to the entire VASG for formal approval; however, such approval has not yet been received.

*VASG-Approved* This status indicates that the ISAC has processed the report as in the “ISAC-Approved” status; additionally, the VASG has reached consensus that the recommendations recorded under “VASG-ISAC Interpretation for IEEE Std 1076-1987” are its official interpretation of IEEE Std 1076-1987.

*Superseded* The ISAC has determined that this issue is identical to, a subcase of, or is entirely resolved by another issue report. The superseding issue is recorded in the “Related IRs” and in the “VASG-ISAC Analysis and Rationale,” “VASG-ISAC Interpretation for IEEE Std 1076-1987,” and “VASG-ISAC Recommendation for Future Revisions” sections, as appropriate.

**Date Submitted:** This field contains the date that the issue was first entered into the VASG-ISAC Issue Report data base. It is presented in dd-mmm-yyyy format. This field is maintained by the ISAC.

**Date Last Revised:** This field contains the date that the issue was last modified by the ISAC. It is presented in dd-mmm-yyyy format. This field is maintained by the ISAC.

Note that the issue report sources, as maintained by the VASG, contain additional fields.

## Description of the Problem

In this section, the author of the issue report defines and illustrates the issue, presenting passages from the standard and other examples as appropriate.

## Proposed Resolution

In this section, the issue author presents his or her preferred resolution of the issue.

## VASG-ISAC Analysis and Rationale

The substance of the deliberations of the ISAC on the issue is recorded in this section. It will typically contain LRM references and examples that illustrate and define the scope of the issue, which upon analysis is sometimes found to be larger than initially reported by the author.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

Once the ISAC has reached consensus on the proper resolution of the issue, the resolution is recorded in this section. Often, alternatives are discussed; if so, one preferred solution is recommended. When the status of the issue is “VASG-Approved,” this recommendation is the official interpretation of IEEE Std 1076-1987.

## VASG-ISAC Recommendation for Future Revisions

The charter of the ISAC is to maintain IEEE Std 1076-1987 by identifying and recommending minimal changes to the language to repair its deficiencies. However, during its deliberations, the ISAC often determines that more pervasive language changes are felt to be required in order to fully address the issue at hand. These suggestions are recorded in this section.

*Note that this section is not part of the official interpretation of the issue, and there is no guarantee that the suggested changes, if any, will ever appear in any version of the VHDL standard.*

## Glossary

This glossary contains brief definitions for the various terms and acronyms used in these interpretations. Terms defined in IEEE Std 1076-1987 are not redefined here.

**1076/B.** The second working draft of IEEE Std 1076-1987. This draft was issued in December 1986 and was subsequently modified to create 1076/C.

**1076/C.** The third, balloted draft of IEEE Std 1076-1987, issued in May 1987. During the negative ballot response, modifications were made to 1076/C to form IEEE Std 1076-1987.

**1076.** The serial number of the PAR for the VASG's VHDL standardization request.

**1076B.** See 1076/B.

**BNF.** Backus-Naur Form or Backus Normal Form. A language for describing the syntax of languages. A variant of BNF is used to describe the syntax of VHDL in the standard; it is also extensively used to illustrate examples in this document.

**CLSI.** CAD Language Systems, Inc. A US-based company. CLSI was funded by the US Air Force to provide technical assistance to the VASG in developing IEEE Std 1076-1987.

**DASS.** The Design Automation Standards Subcommittee of the DATC. The technical organization responsible for all standards-making activities in the DATC.

**DATC.** The Design Automation Technical Committee of the IEEE. An IEEE committee that sponsors, at least in part, all IEEE standards relating to design automation.

**HDL.** Hardware Description Language. A general class of computer languages whose domain is not the description of algorithms to be executed by computer systems, but the description of the computing systems themselves.

**IEEE.** The Institute of Electrical and Electronics Engineers.

**IR.** Issue Report. The formal mechanism whereby the VASG and the ISAC collect, analyze, distribute, and recommend resolutions to issues identified in IEEE Std 1076.

**ISAC.** The Issues Screening and Analysis Committee of the VASG. A group of technical experts that collects and analyzes issues relating to IEEE Std 1076 and recommends solutions to the VASG.

**ISO.** International Organization for Standardization. An international standards-making body.

**LRM.** Language Reference Manual. The IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1987.

**PAR.** Project Authorization Request. The mechanism whereby authorization to develop a standards document in the IEEE is requested and received. The IEEE requires that all of its standardization efforts receive authorization, to avoid duplication and inappropriate efforts.

**TBD.** To be determined, or to be done.

**VASG-ISAC.** The Issues Screening and Analysis Committee of the VASG. A group of technical experts that collects and analyzes issues relating to IEEE Std 1076 and that recommends solutions to the VASG.

**VASG.** The VHDL Analysis and Standardization Group of the DASS. The group responsible for the creation, maintenance, interpretation, and revision of IEEE Std 1076.

**VHDL.** VHSIC Hardware Description Language. The common name for IEEE Std 1076. VHDL was developed from an earlier language, VHDL 7.2, whose development was funded by the US Department of Defense VHSIC Program Office.

**VHSIC.** Very High-Speed Integrated Circuits. Used largely in reference to the VHSIC program. The VHSIC Program Office of the US Department of Defense funded the development of an antecedent of VHDL.





|                      |                                                                                                                                                                  |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Issue Report:</b> | <b>1</b>                                                                                                                                                         |
| Classification:      | <b>Language Definition</b>                                                                                                                                       |
| Summary:             | <i>It is not clear whether array operands for logical operators must be of the same base type. In fact, operand and result types are not spelled out at all.</i> |
| Related IRs:         | None                                                                                                                                                             |
| Relevant Sections:   | 7.2.1                                                                                                                                                            |
| Key Words/Phrases:   | BOOLEAN, logical operator, one-dimensional array type                                                                                                            |
| Current Status:      | VASG Approved                                                                                                                                                    |
| Date Submitted:      | 9 September 1990                                                                                                                                                 |
| Date Last Revised:   | 22 October 1990                                                                                                                                                  |

---

## Description of the Problem

Section 7.2.1, Logical Operators, says:

They are also defined for any one-dimensional array type whose element type is BIT or BOOLEAN...the operands must be arrays of the same length,...and the result is an array with the same index range as the left operand.

There is no explicit statement that the operands for the binary (dyadic) operators must be of the same type. The wording of the quotation above is open to interpretation either way. The accompanying truth tables are also ambiguous with regard to any required type correspondences.

In fact, Section 7.2.1 does not even explicitly state that for simple scalar operands, both operands must be of the same type: either BOOLEAN or BIT. This leaves open the possibility of expressions such as the following:

|                    |                          |
|--------------------|--------------------------|
| '1' or TRUE        | —Valid VHDL expressions? |
| '0' /= '1' and '0' |                          |

Furthermore, since Section 7.2.1 does not mention result types, it can only be inferred that the result type of a logical operation is the same as the type of the operand(s). However, if the above expressions were valid, it is not clear what the result type would be.

In Ada, the type correspondence requirements are made quite explicit. Section 4.5.1 of the Ada LRM starts off:

The following logical operators are predefined for any Boolean type and any one-dimensional array type whose components are of a Boolean type; in either case the two operands have the same type.

This is followed by the usual table with columns for operator, operation, operand type, and result type. This table makes it clear that the result is always of the same type as the operands.

## Proposed Resolution

In our view, it is necessary for both operands of a binary logical operator to be of the same (base) type. If this were not so, then it would not be clear what should be the type of the result. The only two possibilities would be:

- (1) The result type is the same as the type of the left operand
- (2) The result must be determinable from context

Neither of these strikes us as very desirable.

We see no reason not to follow the example of Ada on this issue. Section 7.2.1 of the VHDL LRM should be interpreted as if it had explicitly stated that in all cases, both operands must have the same type. It follows from this that the result is of the same type, but it would be nice if there had been the usual table, with entries such as the following:

| Operator | Operation   | Operand type                                               | Result type                                          |
|----------|-------------|------------------------------------------------------------|------------------------------------------------------|
| and      | Conjunction | BOOLEAN<br>BIT<br>Any array of BOOLEAN<br>Any array of BIT | BOOLEAN<br>BIT<br>Same array type<br>Same array type |

## VASG-ISAC Analysis and Rationale

This is a problem. The solution is relatively straightforward.

The paragraph starting “The following logical operators ...” needs to be rewritten to state explicitly that both operands of a binary logical operator must be of the same type. For example:

The logical operators **and**, **or**, **nand**, **nor**, **xor**, and **not** are defined for predefined types BIT and BOOLEAN. They are also defined for any one-dimensional array type whose element type is BIT or BOOLEAN. For the binary operators **and**, **or**, **nand**, **nor**, and **xor**, the operands must both be of the same type. Furthermore, in the case of the one-dimensional arrays, they must be arrays of the same length, the operation is performed on matching elements of the arrays, and the result is an array with the same index range as the left operand. For the unary operator **not**, the operation is performed on each element of the operand, and the result is an array with the same index range as the operand.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

Read the paragraph “The following logical operators ...” as the paragraph shown in the analysis and rationale section above.

## VASG-ISAC Recommendation for Future Revisions

- (1) Change the paragraph “The following logical operators ...” to the paragraph shown in the Analysis and Rationale section above.
- (2) Add a table that explicitly states each valid operator/operand(s)/result combination.

**NOTE:** There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987.

|                      |                                                                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Issue Report:</b> | <b>2</b>                                                                                                                                   |
| Classification:      | <b>Typographical and Editorial Errors</b>                                                                                                  |
| Summary:             | <i>Correction of a variety of typographical/editorial errors.</i>                                                                          |
| Related IRs:         | 120                                                                                                                                        |
| Relevant Sections:   | 1.2.2, 1.3, 2.2, 2.3, 3.2.1.1, 3.3.1, 4.3.1.2, 4.4, 7.3.2.2, 7.3.5, 9.6.1, 9.7, 12.2, 12.2.1, 12.2.2, 12.2.4, 12.3.1.3, 13.4.2, Appendix B |
| Key Words/Phrases:   | Not applicable                                                                                                                             |
| Current Status:      | VASG Approved                                                                                                                              |
| Date Submitted:      | 18 October 1988                                                                                                                            |
| Date Last Revised:   | 30 July 1990                                                                                                                               |

## Description of the Problem

There are a number of typographical and editorial errors in the LRM (see below).

## Proposed Resolution

These should be corrected.

## VASG-ISAC Analysis and Rationale

Indeed these are typographical and editorial errors and should be corrected.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

- (1) The line in the example from Section 9.6.1 on page 9-12 that reads

```
component COMP port(A,B : inout BIT);
```

should read

```
component COMP port(A,B : inout BIT); end component;
```

This corrects illegal VHDL syntax.

- (2) The lines in the example from Section 1.2.2 on page 1-8 that read

```
component Full_Adder
port(X, Y, Cin: Bit; Cout, Sum: out Bit);
```

should read

```
component Full_Adder
  port(X, Y, Cin: Bit; Cout, Sum: out Bit);
end component;
```

This corrects illegal VHDL syntax.

- 
- (3) The lines in the example from Section 2.3 on page 2-6 that read

```
Check (Setup=>10ns, D=>Bus, C=>Clk1);  
Check (Hold=>5ns, D=>Bus, C=>Clk2);  
Check (15ns, Bus, Clk);      —ambiguous if Data'Base = Clock'Base
```

should read

```
Check (Setup=>10 ns, D=>Wire, C=>Clk1);  
Check (Hold=>5 ns, D=>Wire, C=>Clk2);  
Check (15 ns, Wire, Clk);      —ambiguous if Data'Base = Clock'Base
```

This corrects illegal use of a VHDL reserved word.

- 
- (4) The line from Section 7.3.5 on page 7-12 that reads

```
type_conversion ::= type_mark ( expression )
```

should be indented to match the other grammar productions appearing in the LRM.

- 
- (5) In the glossary (Appendix B) on page B-1, the entries for “alias” and for “aggregate” appear in the wrong order. The order of these two entries should be reversed.

- 
- (6) The line in the example from Section 1.3 on page 1-10 that reads

```
for StructureView
```

should read

```
for Structure_View
```

This is a typographical error.

- 
- (7) The line in the example from Section 3.3.1 on page 3-15 that reads

```
variable NEXT : LINK := HEAD.SUCC;
```

should read

```
variable NXT : LINK := HEAD.SUCC;
```

This corrects illegal use of a VHDL reserved word.

- (8) The line in the example from Section 4.4 on page 4-14 that reads

**type POSITIVE is INTEGER range 1 to INTEGER'HIGH;**

should read

**subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;**

This corrects illegal VHDL syntax.

- 
- (9) The line in the example from Section 9.7 on page 9-14 that reads

**B : block**

should read

**Gen : block**

Similarly, the line in the same example that reads

**end block B;**

should read

**end block Gen;**

These corrections allow the signal B to be visible where it is referenced in example.

- 
- (10) The last two paragraphs of Section 2.2 on page 2-5 are notes but are not identified as such. These two paragraphs should be preceded by the header *Note*:

- 
- (11) The last sentence of Section 12.3.1.3 on page 12-4, which reads

Elaboration of a size constraint consists of the evaluation of the expression.

should be deleted. Size constraints have been removed from the language.

- 
- (12) Item 6 in Section 7.3.2.2 on page 7-11 that reads

6. As the expression defining the initial value of the drivers of one or more signals in an initialization specification, where the corresponding subtype is a constrained array subtype;

should read

6. As the expression defining the default values of signals in a signal declaration, where the corresponding subtype is a constrained array subtype;

---

(13) The definition of “resume” in the glossary reads

A process that resumes in a given simulation cycle becomes ready to execute and will execute at the end of the given simulation cycle.

but should read

A process that resumes in a given simulation cycle becomes ready to execute and will execute at the end of the given simulation cycle.

This is a typographical error.

---

(14) The sentence in the glossary from the definition of “specification” that reads

There are four kinds of specifications: attribute specifications, initialization specifications, configuration specifications, and disconnection specifications.

should read

There are three kinds of specifications: attribute specifications, configuration specifications, and disconnection specifications.

---

(15) There are two definitions of “guarded signal” in the glossary. The second of these is more informative, and thus the first definition should be removed.

---

(16) The sentence immediately above the examples in Section 13.4.2 on page 13-5 that reads

An exponent indicates the power of the base by which the value of the based literal with the exponent.

is not well formed. It should be replaced with

An exponent indicates the power of the base by which the value of the based literal without the exponent is to be multiplied to obtain the value of the based literal with the exponent. An exponent for a based integer literal must not have a minus sign.

---

(17) The references to “generic map clause” and “port map clause” in the seventh, eighth, ninth, and tenth items in the list beginning in Section 3.2.1.1 on page 3-11 should be read “generic map aspect” and “port map aspect,” respectively.

- (18) The four references to “port map clause” in the paragraph beginning “If a subelement of a resolved signal of a composite type is ...” from Section 4.3.1.2 on page 4-6 should be read “port map aspect.”

- (19) In Sections 12.2, 12.2.1, 12.2.2, and 12.2.4 on page 12-2, the references to “generic map clause” and “port map clause” should be read “generic map aspect” and “port map aspect,” respectively. There are two such references in 12.2, two in 12.2.1, one in the title of 12.2.2, one in the body of 12.2.2, one in the title of 12.2.4, and one in the body of 12.2.4.

- (20) The line in the example from Section 1.3.2 on page 1-13 that reads

**use entity** StdCells.PadTriState4 (StdCells.DataFlow)

should read

**use entity** StdCells.PadTriState4 (DataFlow)

This corrects illegal VHDL syntax.

- (21) The sentence in Section 3.1.3.1 on page 3-7 that reads

The only predefined physical is type TIME.

should read

The only predefined physical type is TIME.

This is a typographical error.

- (22) The sentence in Section 3.1.3.1 on page 3-7 that reads

The range of TIME is implementation-dependent, but it is guaranteed to include the range -2147483647 to +2147483647.

should read

The range of TIME is implementation-dependent, but it is guaranteed to include the range -2147483647 to +2147483647.

The writing of “46” for “64” is a typographical error.

- (23) The literals “-1E38” and “+1E38” in Section 3.1.4.1 on page 3-8 should be read “-1.0E38” and “+1.0E38,” respectively. Similarly, the literals “-1E38” and “+1E38” in Section 3.1.4 on page 3-8 should be read “-1.0E38” and “+1.0E38,” respectively. These changes make the type of these literals `universal_real`, which is the appropriate type for specifying the

minimum bounds of floating point types.

---

(24) The list item in Section 4.3 on page 4-3 that reads

o a formal port of a design entity.

should read

o a formal port.

The phrase “of a design entity” mistakenly prevents the list item from applying to block ports.

---

(25) The line from the example in Section 3.3 on page 3-14 that reads

**type** BUFFER\_PTR **is access** BUFFER;

should read

**type** BUFFER\_PTR **is access** MY\_BUFFER;

This corrects illegal use of a VHDL reserved word.

---

(26) The example in Section 1.3.2 on page 1-13 that reads

**for** D1: DSP

*—binding specified in design entity or else defaults*

**for** Filterer

*—configuration items for filtering components*

**end for;**

**for** Processor

*—configuration items for processing components*

**end for;**

**end for;**

should read

**for** D1: DSP

*—binding specified in design entity or else defaults*

**for** DSP\_STRUCTURE

**for** Filterer

*—configuration items for filtering components*

**end for;**

**for** Processor

*—configuration items for processing components*

**end for;**

**end for;**

**end for;**

This corrects a missing block configuration.

## VASG-ISAC Recommendation for Future Revisions

Correct these problems in the standard.

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

**Issue Report: 3**

Classification: **Examples, Notes, and Appendixes**

Summary: *Examples in the standard do not have a space between the abstract literal and the unit name of physical literals.*

Related IRs: None

Relevant Sections: 3.1.3, 13.2

Key Words/Phrases: lexical element, physical literal

Current Status: VASG Approved

Date Submitted: 31 October 1988

Date Last Revised: 30 July 1990

---

## Description of the Problem

The fourth paragraph of Section 13.2 of the LRM states that:

... At least one separator is required between an identifier or an abstract literal and an adjacent identifier or abstract literal.

Thus, a space is always required between the abstract literal and the unit name in a physical literal. Nevertheless, many of the examples in the LRM incorrectly include physical literals that contain no space between the abstract literal and the unit name.

### Example

—LRM 10.3

—Comments have been deleted

```
L1: block
    signal A,B: Bit;
begin
L2: block
    signal B: Bit;
    begin
        A<=B after 5ns;
        B<=L1.B after 10ns;
    end block;
    B<=A after 15ns;
end block;
```

## Proposed Resolution

Yes indeed, spaces are required between the abstract literal and the unit name. The examples are in error and should be changed.

---

## VASG-ISAC Analysis and Rationale

In this case the standard is very clear that a space is required. Unfortunately, the examples in the LRM and examples in several other documents, such as the VHDL tutorial, do not contain spaces. This has caused many users (and some implementors) not to use (require) spaces. The standard could be changed to allow this by relaxing the restriction that a separator is required between an

abstract literal and an adjacent identifier. However, this causes problems when scanning a decimal literal such as 10E2. Is this the decimal literal whose value is 1000 or a physical literal whose value is 10E2s? Current implementations resolve this problem by always deciding in favor of the decimal literal when this situation occurs. These implementations only require a separator when the unit name of the physical literal starts with an “E.” Although this works, in the sense that it will parse the incorrect examples, it is certainly not as clean as always requiring a separator. Since 10 ns seems as readable as 10ns, there is no good reason to change the standard.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

Clean up the examples so as to have a space between the abstract literal and the unit name in physical literals always.

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

### Issue Report: 3

#### Formal Comments

The resolution to this issue is not what was intended, either in Section 7.2 or in IEEE Std 1076-1987. The quoted sentence from Section 13.2 is a holdover from Ada and is intended to ensure that, e.g., “x1234” is treated as one identifier and not an identifier followed by an integer. The quoted sentence establishes four cases:

- (1) Identifier followed by identifier;
- (2) Identifier followed by numeric literal;
- (3) Numeric literal followed by numeric literal;
- (4) Numeric literal followed by identifier.

Case 4 never occurs in Ada. The only case in Ada where a group of digits can precede a letter is in the case of numeric literals with an exponent; there, of course, no separator is allowed.

Unlike Ada, VHDL has physical literals, where an identifier can follow a number. While it is true that a naive lexer can be confused by “10E,” where E is a physical literal, such naive lexers are insufficient to process VHDL, even if this requirement is imposed. VHDL lexers must distinguish between character literals and apostrophes; e.g., between ‘A’ and A'B'C. (A character literal is a single token, while an apostrophe is also a token in its own right.) The same look-ahead required to distinguish between character literals and apostrophes can be used to distinguish between “10E” and “10E4.” (Note that Ada does not suffer from this problem either, since Ada has only predefined attributes, none of which have single-character names.)

Thus, it is reasonable to relax this restriction.

|                                                                                            |
|--------------------------------------------------------------------------------------------|
| NOTE: These formal comments are not part of the official interpretations of this document. |
|--------------------------------------------------------------------------------------------|

## Issue Report:

4

Classification:

### Language Definition

Summary:

*It is unclear how to determine the resolution function applicable to a signal under certain circumstances.*

Related IRs:

None

Relevant Sections:

4.2, 4.3.1.2

Key Words/Phrases:

resolution functions, subtype indications

Current Status:

VASG Approved

Date Submitted:

3 February 1989

Date Last Revised:

28 January 1991

---

## Description of the Problem

Section 4.3.1.2 (on page 4-5) of the LRM defines a resolved signal in the following manner:

If the name of a resolution function appears in the declaration of a signal, or in the declaration of the subtype used to declare the signal, then that resolution function is associated with the declared signal. Such a signal is called a *resolved signal*.

Moreover, Section 4.2, Paragraph 4 (on page 4-3) defines the meaning of a subtype indication including a resolution function:

If a subtype indication includes a resolution function name, then any signal declared to be of that subtype will be resolved, if necessary, by the named function (see Section 2.4). A resolution function name has no effect on the declaration of objects other than signals or on the declarations of files, aliases, attributes, or other subtypes.

The determination of the resolution function of a resolved signal is clear in most cases. Consider the following examples:

— *Example 1:*  
**function** F (V: Bit\_Vector) **return** Bit;  
**subtype** Wired\_Bit **is** F Bit;  
**subtype** Local\_Bit **is** Wired\_Bit;  
  
**signal** S1: Wired\_Bit;  
**signal** S2: F Bit;  
**signal** S3: Local\_Bit;

In this example, S1 and S2 clearly are resolved signals. It also seems clear that S3 is not a resolved signal, since there is neither a resolution function appearing in the declaration of S3, nor in the declaration of the subtype used to declare S3 (Local\_Bit).

Even though the LRM is clear, is the situation with respect to S3 desirable? Subtypes, in both VHDL and ADA, are used to supply additional constraints on the type mark. Provided that one considers the imposition of a resolution function a constraint, the declaration of Local\_Bit in Example 1 seems to *remove* a constraint from Wired\_Bit. By way of analogy, see the next example:

— *Example 2:*  
**subtype** Byte **is** Integer **range** -128 to 127;  
**subtype** Char **is** Byte;

Clearly, Char'Low = Byte'Low and Char'High = Byte'High, and Char'Low /= Integer'Low and Char'High /= Byte'High. Resolution functions should be “passed through” in similar fashion.

There is also an ambiguity in the LRM with respect to resolution functions. It is best illustrated with another example:

— *Example 3:*  
**function** F (V: Bit\_Vector) **return** Bit;  
**function** G (V: Bit\_Vector) **return** Bit;  
  
**subtype** Wired\_Bit **is** F Bit;  
**signal** S: G Wired\_Bit;

In Example 3, which resolution, F or G, is used to resolve the value of S? Note that this ambiguity exists no matter what interpretation is selected for S3 of Example 1; however, if my suggested interpretation is accepted, the effects of this ambiguity are more pervasive.

## Proposed Resolution

I suggest that my interpretation of S3 in Example 1 be adopted, and that the ambiguity of Example 3 be resolved in favor of the function G. To implement, I suggest that the definition of a resolved signal in Section 4.3.1.2 be changed to the following:

If the name of a resolution function appears in the declaration of a signal, or in the declaration of the subtype used to declare the signal, or (recursively) in the definition of any subtype used to declare the subtype, then the first resolution function encountered, when examining the declarations in this order, is associated with the signal.

A signal that has a resolution function associated with it is called a *resolved signal*.

---

## VASG-ISAC Analysis and Rationale

As is pointed out above, there are two issues here. These issues are treated individually as follows.

The first issue concerns whether or not S3 in Example 1 is a resolved signal. As suggested above, the standard seems to make it clear that S3 is not a resolved signal. The key here is the phrase “... or other subtypes” in the quoted sentence from Section 4.2. It should be observed that some have suggested the sentence from Section 4.2:

If the subtype indication does not include a constraint, the subtype is the same as that denoted by the type mark

indicates that Local\_Bit and Wired\_Bit are the “same” subtype and hence both should either imply or not imply a resolution function. However, this would lead to the situation where whether or not the resolution function was “passed through” would depend on the existence of a constraint, and this seems like an unacceptable interpretation. Hence, the line of reasoning that suggests that S3 is not a resolved signal is preferred. The submitter of this IR questions whether this is “desirable.”

This is clearly subjective and depends on whether it is natural to think of the resolution-function property of a subtype like a constraint. Without taking a position on this, there seems to be little support in the present LRM for treating the resolution-function property of a subtype like a constraint, and, therefore, S3 in Example 1 should be considered a nonresolved signal.

The second issue deals with Example 3, where it is clear that S is a resolved signal, but it seems ambiguous which resolution function resolves S. There seem to be three choices: make this case an error, use F, or use G. Using F seems counterintuitive given the presence of G at the site of the declaration of S (i.e., the appearance of F is “further removed” from the declaration of the signal in question). There is no suggestion in the standard that this case should be treated as an error, so the interpretation that S is resolved by G is preferred. This interpretation is also more consistent with the recommendation on the previous issue that the resolution-function property of a subtype not be treated like a constraint.

## **VASG-ISAC Interpretation for IEEE Std 1076-1987**

Signal S3 in Example 1 is not a resolved signal. That is, the resolution-function property of a subtype does not behave like a subtype constraint (and does not “pass through” from one subtype declaration to the next).

Signal S in Example 3 is a resolved signal that is resolved with G. That is, the applicable resolution function can be “overridden” at the point of the signal declaration.

## **VASG-ISAC Recommendation for Future Revisions**

Consider whether the resolution-function property should be a distinct kind of subtype characteristic or whether the semantics of subtype resolution-function properties should correspond to subtype constraint semantics.

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

**Issue Report: 5**

Classification:

**Language Definition**

Summary:

*Visibility of architecture name in the entity aspect of a binding indication is not well defined.*

Related IRs:

None

Relevant Sections:

5.2.1.1, 10.3, 11.2

Key Words/Phrases:

binding indication, configuration, libraries, visibility

Current Status:

VASG Approved

Date Submitted:

3 February 1989

Date Last Revised:

1 October 1990

---

**Description of the Problem**

The visibility of the architecture name in an entity aspect of a binding indication is not well defined. Section 5.2.1.1 (pages 5-4 and 5-5) seems to imply that the architecture exists and is visible at the time the binding is analyzed, but this aspect of visibility is not covered in Section 10.3. In particular, this section does not define the name of an architecture to be visible outside of itself, either directly or by selection, except in the case of a block specification in a block configuration for an architecture (Item 2 on page 10-3).

It seems that the entity must exist, else the generic map aspect and the port map aspect of the binding indication could not be checked for errors during analysis; is this error analysis in fact required?

It also seems that the architecture must exist if the architecture is configured by the current design unit, else the existence of these blocks could not be verified; is this verification in fact required?

**Proposed Resolution**

In Section 5.2.1.1, explicitly state that the architecture must exist during the analysis of the design unit containing the entity aspect.

In Section 10.3, add an item between Items 2 and 3 at the bottom of page 10-3:

- 2a. For an architecture body associated with a given entity name: at the place of an architecture name (between the parentheses) in the first form of an entity aspect in a binding indication.

---

**VASG-ISAC Analysis and Rationale**

This IR is deceptive in the sense that it appears simple but exposes several rather fundamental differences in the way VHDL is supported and in the way it is used.

Three different views of VHDL configuration information have been implemented and are currently provided by vendors. For descriptive purposes herein, these are referred to as SEMI-soft, HARD, and HARD-EVEN-IN-ARCHITECTURES. A fourth view of configuration information, SOFT, has not been implemented (to the knowledge of the ISAC), but is conceptually very appealing. For the purpose of explaining these differing views, consider the following VHDL:

```

entity one is end one;
architecture a of one is
  component comp end component;
begin
  l:comp;
end a;

entity two is end two;
architecture a of two is
  component three end component;
begin
  l:three;
end a;

entity three is end three;
architecture a of three is
begin
  ...
end a;

use work.all;
configuration c of one is
  for a
    for l:comp use entity two(a);
  end for;
  end for;
end c;

```

Also, for the purpose of this discussion, a design unit **DEPENDS ON** another design unit if reanalyzing the latter unit requires reanalyzing the former unit before resimulating.

For any of the above-mentioned views, the configuration above analyzes and elaborates correctly. With the SEMI-SOFT view, the configuration does not **DEPEND ON** architecture a of two (or on architecture a of three) but it does **DEPEND ON** architecture a of one. In both the HARD and HARD-EVEN-IN-ARCHITECTURES views, the configuration **DEPENDS ON** all three architectures. In the SOFT view, the configuration does not **DEPEND ON** any of these architectures.

These differences with respect to dependencies give rise to differences concerning when semantic analysis is performed and when errors are detected. For SOFT, very little (if any) semantic analysis is performed during analysis of the configuration. Error messages for undeclared identifiers, etc. are not seen by the user until elaboration time. FOR SEMI-SOFT, semantic errors in the “explicitly written” part of the configuration are reported at analysis time of the configuration; semantic errors with respect to the implicit aspects of the configuration are not reported until elaboration time. For HARD and HARD-EVEN-IN-ARCHITECTURES, all semantic errors (for both explicit and implicit VHDL) are reported at analysis time of the configuration.

The difference between HARD and HARD-EVEN-IN-ARCHITECTURES can be seen by considering the following VHDL:

```

entity five is end five;
architecture a of five is
begin
    ...
end a;

entity four is end four;
architecture a of four is
    component c end component;
    for l:c use entity work.five(a);
begin
    l:c;
end a;

```

In each of the SOFT, SEMI-SOFT, and HARD views of VHDL, architecture a of four above does not **DEPEND ON** architecture a of five. In the fourth view, **HARD-EVEN-IN-ARCHITECTURES**, this dependency does exist.

How could this result in these different interpretations of VHDL configuration information? The proponents for the various views appear to feel the standard supports (or at least allows) their particular views. The ISAC can reach no agreement on which of these views (if any) the standard does require. The most relevant LRM references here appear to be the last paragraph of 11.4, the last paragraph of 1.3.2, most of 5.2.1.1 and 5.2.2. A main source of the ambiguity seems to be lack of a clear distinction in the standard between analysis- time and elaboration-time errors.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

After much discussion, the ISAC has concluded there is no good way to solve this problem in the short term. It does not appear to be practical or desirable to require any of these views to be supported by all VHDL implementations in the near future. The ISAC recommendation is for the VASG simply to point out to VHDL users that these differences exist in VHDL implementations. Market pressures will encourage implementors to provide the support that offers the best functionality for users. It may make sense for future versions of VHDL to clarify the semantics of configuration information to the point of narrowing (perhaps to one) the allowed views. It is not clear there is a single “best” interpretation; perhaps some user-selectable control over the interpretation of the configuration information is the best answer. Until then, users should be warned about the different implementation strategies that exist. A suggestion to lessen interoperability problems when sending VHDL source would be to assemble the design units in the canonical order that is acceptable to any of the known views. This is the ordering defined by the **DEPENDS ON** requirements of the **HARD-EVEN-IN-ARCHITECTURES** view. It is true that not all VHDL descriptions have such a canonical ordering (recursively instantiated architectures).

Thus, in terms of the proposed resolution, the first aspect of the proposal is rejected. An implementation may or may not require the architecture to exist during the analysis. The intent behind the second aspect of the recommendation is accepted. The Chapter 10 visibility rules should be complete and cover the case of the architecture body identifier in an entity aspect of a binding indication.

## Issue Report:

7

Classification:

## Language Definition

Summary:

*The semantics of component configuration are in error.*

Related IRs:

None

Relevant Sections:

1.3.2

Key Words/Phrases:

component configuration

Current Status:

VASG Approved

Date Submitted:

4 February 1989

Date Last Revised:

28 January 1991

## Description of the Problem

Paragraph 3 of Section 1.3.2 (on page 1-12) reads:

It is an error if both an explicit configuration specification (in an architecture body) and a component configuration containing a binding indication (in a configuration declaration) apply to the same component instance.

The intent of this paragraph is to ensure that there are not multiple (and possibly conflicting) bindings applicable to a given component instance.

However, it is possible to supply an explicit configuration specification that does not imply any design entity, in which case, if a nondefaulted binding of the component to a design entity is to occur, a component configuration containing a binding indication is necessary. This case is best illustrated with an example:

```
architecture A of E is
  component C
  end component;
```

```
  for all:C use open;      —does not imply a design entity
begin
  ...
end A;
```

```
configuration C of E is
  for A
    for all:C
      use entity CE (CA);
    ...
  end for;
end for;
end C;
```

## Proposed Resolution

Modify the above paragraph from the LRM to read:

It is an error if both an explicit configuration specification (in an architecture body) and a component configuration containing a binding indication (in a configuration declaration) apply to the same component instance unless the entity aspect of the configuration specification is **open**.

---

## **VASG-ISAC Analysis and Rationale**

The VHDL given above should be considered illegal. The proposed resolution is not recommended since the present standard seems clear on this matter and the proposed resolution adds language complexity for relatively little gain. Furthermore, the suggestion would be further complicated if it considered the similar situation where a design entity was implied but the architecture body was not implied. Finally, the proposed resolution can be viewed as a way of “overriding” the “assertion” in the architecture that the instance is open. This is not consistent with the language philosophy behind configuration declarations.

## **VASG-ISAC Interpretation for IEEE Std 1076-1987**

The standard should be taken literally with regard to this issue, and an alternate interpretation is not recommended.

## **VASG-ISAC Recommendation for Future Revisions**

There does not seem to be compelling reason to revisit this issue when the next version of the standard is being prepared.

|                      |                                                       |
|----------------------|-------------------------------------------------------|
| <b>Issue Report:</b> | <b>9</b>                                              |
| Classification:      | <b>Examples, Notes, and Appendixes</b>                |
| Summary:             | <i>The glossary entry for “Generic” is incorrect.</i> |
| Related IRs:         | None                                                  |
| Relevant Sections:   | Appendix B                                            |
| Key Words/Phrases:   | generic                                               |
| Current Status:      | VASG Approved                                         |
| Date Submitted:      | 4 February 1989                                       |
| Date Last Revised:   | 10 October 1990                                       |

---

## Description of the Problem

The glossary entry for “generic” states, in part:

A generic is a constant declared in a component declaration or an entity declaration.

However, generics can also occur in the block header of block statements.

## Proposed Resolution

Modify the above sentence to read:

A generic is a constant declared in the block header of a block statement, a component declaration, or an entity declaration.

---

## VASG-ISAC Analysis and Rationale

Yes, generics can occur in the block header of a block statement.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

The recommended rewording of the glossary should be assumed.

## VASG-ISAC Recommendation for Future Revisions

The recommended rewording of the glossary should be included.

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

**Issue Report: 10**Classification: **Language Definition**Summary: *The disconnection time of a guarded signal is unclear when the signal is associated through a port map.*

Related IRs: 63

Relevant Sections: 5.3, 9.5, 12.3.2.3

Key Words/Phrases: disconnection specification, guarded signals, interface signals, signal association

Current Status: VASG Approved

Date Submitted: 9 November 1988

Date Last Revised: 11 January 1991

**Description of the Problem**

It appears that more than one disconnection specification is applicable to a signal when the signal is associated through a port map. Consider the following example:

```

B1: block
    function ResFn (V: Bit_Vector) return Bit is ... end ResFn;
    signal S1: ResFn Bit bus;
    disconnect S1: Bit after 10 nS;
begin
B2: block (Ena = '1')
    port (S2: out ResFn Bit bus);
    port map (S2 => B1.S1);
    —disconnect S2: Bit after 0 nS;    —implicit disconnection specification
begin
    S2 <= guarded '1' after 42 nS;
end block B2;
end block B1;

```

In the equivalent process statement of the guarded signal assignment statement, what turnoff time is used? Is the explicitly specified time for B1.S1 (10 nS) used, or is the implicitly specified time for B2.S2 used (0 nS)?

(If an explicit disconnection specification for S2 was given, and the time was other than 10 nS, the situation would be even more confusing.)

Section 12.3.2.3, item 3 on page 12-6 states:

The disconnection time is associated with each affected signal for later use in constructing disconnection statements in the equivalent processes for guarded assignments to the affected signals.

In Section 9.5, paragraph 14 on page 9-8:

... [the] time expression [of the waveform of the sequential signal assignment forming the disconnection statement] is given by the applicable disconnection specification.

Section 5.3, paragraph 8 on page 5-7 states:

It is an error if more than one disconnection specification applies to drivers of the same signal.

However, “applicable disconnection specification” is not defined. Moreover, nowhere in the LRM is the effect of signal association on disconnection times of the associated signals discussed.

## Proposed Resolution

Use as the turnoff time the value supplied by the disconnection specification applicable to the local signal. In the above example, the applicable turnoff time is 0 nS. State this explicitly in Section 5.3.

---

## VASG-ISAC Analysis and Rationale

The standard is clear on this point; furthermore, there are no inconsistencies in the approach of the standard. Disconnection specifications corresponding to a given signal are used only to construct the equivalent processes for guarded signal assignment statements that drive that signal; hence, any disconnection specifications applicable to any signals associated with the signal via association lists are not germane.

To illustrate, here is the process equivalent to the guarded signal assignment statement in the above example:

```
process  
begin  
  if GUARD then  
    S2 <= '1' after 42 nS;  
  else  
    S2 <= null after 0 nS;  
  end if;  
  wait on GUARD;  
end process;
```

Note that S2, and not B1.S1, is driven by the guarded signal assignment statement; therefore, it is the disconnection specification of S2 that applies.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

No changes need to be made to the standard or any interpretations with respect to this issue. However, see IR 63 regarding “applicable disconnection specifications.”

## VASG-ISAC Recommendation for Future Revisions

No changes need to be made to the standard or any interpretations with respect to this issue. However, see IR 63 regarding “applicable disconnection specifications.”

|                      |                                                                |
|----------------------|----------------------------------------------------------------|
| <b>Issue Report:</b> | <b>12</b>                                                      |
| Classification:      | <b>Language Definition</b>                                     |
| Summary:             | <i>Can implicitly declared subprograms be attributed?</i>      |
| Related IRs:         | 14, 17                                                         |
| Relevant Sections:   | 3.4.1, 4.4, 5.1, 6.6, 10.3                                     |
| Key Words/Phrases:   | attribute names, implicit subprograms, user-defined attributes |
| Current Status:      | VASG Approved                                                  |
| Date Submitted:      | 7 February 1989                                                |
| Date Last Revised:   | 1 April 1991                                                   |

---

## Description of the Problem

The LRM is unclear on the semantics of associating user-defined attributes with the implicitly declared subprograms defining the operations on files. (These subprograms are, at present, the only implicitly defined subprograms.) To illustrate, consider the following example:

```

type F is file of Integer;
attribute Implicit: Boolean;
attribute Implicit of READ, WRITE: procedure is True;
attribute Implicit of ENDFILE : function is True;

```

Is this example a legal VHDL fragment? Specifically, do the entity classes “procedure” and “function” include the procedures and the function implicitly declared by the declaration of the file type F?

## Proposed Resolution

Associating user-defined attributes with implicitly declared subprograms should be legal. Either permit the entity class “procedure” to include the implicitly declared procedures and permit “function” to include the implicitly declared functions, or introduce new entity classes (e.g., “implicit procedure” and “implicit function”) to cover these cases.

---

## VASG-ISAC Analysis and Rationale

The request to be able to attribute implicitly declared subprograms is reasonable. Furthermore, it does not appear to cause any conflict or problems with the rest of the language definition.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

Assume that the first paragraph on page 5-2:

The attribute designator must denote an attribute. The entity name list identifies those entities that inherit the attribute, as defined below:

refers to both implicit and explicit subprograms.

## VASG-ISAC Recommendation for Future Revisions

Reword the first paragraph on page 5-2 as follows:

The attribute designator must denote an attribute. The entity name list identifies those entities, *both implicit and explicit*, that inherit the attribute, as defined below:

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

**Issue Report: 13****Classification: Language Definition****Summary:** *Interaction of slice prefix and discrete-range directions is unclear.***Related IRs:** 96**Relevant Sections:** 3.1, 6.5, 7.3.1, 14.1**Key Words/Phrases:** direction, null, slice**Current Status:** VASG Approved**Date Submitted:** 29 May 1987**Date Last Revised:** 13 March 1991

## Description of the Problem

The direction and bounds of a null slice are not clearly specified by the LRM.

## Proposed Resolution

A null slice can occur either if the range is null or if the prefix and discrete range differ in direction. For a null slice due to a null range, both the direction and the bounds of the null slice come from the discrete range. If the prefix and discrete range differ in direction, the direction of the null slice comes from the prefix, and its bounds come from the discrete range.

## VASG-ISAC Analysis and Rationale

The standard states (Section 6.5) that

The bounds of the discrete range define those of the slice and must be of the type of the index of the array. The slice is a *null slice* if the discrete range is a null range, or if the direction of the discrete range is not the same as that of the object denoted by the prefix of the slice name.

The problem is to define the direction of the slice so that a null slice will result with the bounds determined from the discrete range when the direction of the discrete range differs from the direction of the prefix. In fact, this can be impossible, as shown by the following example.

*Example*

*This example shows that it may be impossible to define the direction of a slice so that a null slice results.*

```
begin
process
```

```
  type index_type is (red);
  type int_array is array (index_type range <>) of integer;
```

```
  procedure test (X : int_array) is
  begin
    assert X'length = 0;    —X must be null
```

```

    assert X'left = red;    —Bounds are determined by range red downto red
    assert X'right = red;
end test;

variable A : int_array (red to red) := (red => 1);
begin
    test(A(red downto red));
end process;
```

Note that X is a null array since the direction of the discrete range in the slice A(red downto red) is not the same as the direction of the prefix. On the other hand, the bounds, which are taken from the discrete range, are both red. This is apparently a contradiction since a null range is either an ascending range with  $R < L$  or a descending range with  $L < R$ . Since  $X'left = X'right$ , neither of these cases applies, and one is forced to conclude that X is not a null array. The problem is that there is no way to describe a null range whose bounds come from the type index\_type.

This is also a problem in determining the bounds and directions of bit string literals (see Section 7.3.1). In this case, a null range is specified by making the right bound of the array value be the predecessor of the left bound for ascending ranges and the successor of the left bound for descending ranges. Note that for the index type index\_type this will give an error since there is no predecessor or successor of red. The Ada LRM (Section 4.2, paragraph 3) is very explicit about this.

The evaluation of a null string literal raises the exception CONSTRAINT\_ERROR if the lower bound does not have a predecessor (see 3.5.5).

IEEE Std 1076-1987 leaves this as an implicit consequence of the attempt to evaluate PRED or SUCC when these are not defined.

Since there is no way to describe a null range in the above case, the meaning of the slice A(red downto red) must be defined in some other way.

One alternative that has been suggested is the following: the result of slicing with a discrete range whose direction is opposite from the direction of the prefix reverses the left-to-right order of the elements of the resulting array of the slice. There is currently no operator in the language that accomplishes this “mirroring” operation. However, it is easy to write a function that will reverse the order of the elements of its array argument, so this does not seem like a necessary addition to the language. Since this is not necessary and represents a further complication of the language, it does not seem like a desirable addition to VHDL.

The simplest thing to do is to make it illegal to slice an array with a discrete range whose direction differs from the direction of the array.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

Interpret paragraph 3 of Section 6.5 as follows:

The bounds of the discrete range define those of the slice and must be of the type of the index of the array. The slice is a null slice if the discrete range is a null range. It is an error if the direction of the discrete range is not the same as that of the prefix of the slice name.

## VASG-ISAC Recommendation for Future Revisions

Adopt the wording of the previous section.

The example in this section that now reads

Data(24 **to** 25)                      —*a null slice*

must also be changed, since under the new interpretation, it is an illegal slice, not a null slice. One possibility is

Data(24 **downto** 25)                      —*a null slice*

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

|                      |                                                                             |
|----------------------|-----------------------------------------------------------------------------|
| <b>Issue Report:</b> | <b>15</b>                                                                   |
| Classification:      | <b>Language Definition</b>                                                  |
| Summary:             | <i>Bounds and direction of certain unconstrained formals are undefined.</i> |
| Related IRs:         | None                                                                        |
| Relevant Sections:   | 3.2.1.1                                                                     |
| Key Words/Phrases:   | subelement association, unconstrained formals                               |
| Current Status:      | VASG Approved                                                               |
| Date Submitted:      | 10 February 1989                                                            |
| Date Last Revised:   | 2 April 1991                                                                |

---

## Description of the Problem

Section 3.2.1.1 defines the index range of an unconstrained formal to be taken from the “corresponding association element.” This definition is not applicable if the subelements of the formal are associated on an individual basis. At one time it was the intent of CLSI to disallow subelement associations for an unconstrained formal; it is probably an oversight that this is not.

## Proposed Resolution

Assume that an unconstrained formal cannot have its subelements associated on an individual basis.

---

## VASG-ISAC Analysis and Rationale

The standard does not specifically address the issue of whether an unconstrained formal (or local) can have its subelements associated on an individual basis. Some implementors have inferred that subelement association is allowable in this situation; others have concluded that it is illegal. So, for IEEE Std 1076-1987, the use of such constructs may result in nonportable code.

However, there is agreement that it is reasonable to allow subelement association in these cases. Therefore, language should be added to the next revision to allow this.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

Not all implementations support subelement association for unconstrained formals or locals. Therefore, the use of such an association is nonportable.

## VASG-ISAC Recommendation for Future Revisions

Add appropriate wording to the standard to the effect that subelement association is allowed when the formal (or local) parameters are unconstrained arrays, and that, in such cases, the direction of the range is that of the type of the formal (or local) parameter and that the high and low bounds are determined from the maximum and minimum indexes given in the subelement association.

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

|                      |                                                                |
|----------------------|----------------------------------------------------------------|
| <b>Issue Report:</b> | <b>17</b>                                                      |
| Classification:      | <b>Language Definition</b>                                     |
| Summary:             | <i>Can predefined operators be attributed?</i>                 |
| Related IRs:         | 12, 14                                                         |
| Relevant Sections:   | 3, 4.4, 5.1, 6.6, 7.2, 10.3                                    |
| Key Words/Phrases:   | attribute names, predefined operators, user-defined attributes |
| Current Status:      | VASG Approved                                                  |
| Date Submitted:      | 7 February 1989                                                |
| Date Last Revised:   | 22 March 1991                                                  |

---

## Description of the Problem

The LRM is unclear on the semantics of associating user-defined attributes with the predefined operators (and on functions overloading them). Consider the following example:

```

type T is range 1 to 10;
attribute Predefined: Boolean;
attribute Predefined of
    "=", "/=", "+", "-", "*", "/", "mod", "rem", "abs", "**": function
is True;

```

This is syntactically allowed; the entity designator of an attribute specification may be either a simple name or an operator symbol. However, is the entity class **function** appropriate to these predefined operators? The LRM is unclear; in some places (e.g., when considering named association), the LRM maintains a distinction between predefined operators and user-defined functions, even when those functions overload a predefined operator. (For example, a user-defined “+” function may be called with named association; the predefined operator with the same parameter and result-type profile may not.) In Section 5.1, the LRM seems to meld the concepts.

## Proposed Resolution

Explicitly state in Section 5.1 that the entity class **function** applies both to user-defined functions and predefined operators. Alternatively, introduce a new entity class, **operator**, that specifically designates predefined operators.

---

## VASG-ISAC Analysis and Rationale

This is very similar to IR 12. The same interpretation shall be followed.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

Assume that the first paragraph on page 5-2:

The attribute designator must denote an attribute. The entity name list identifies those entities that inherit the attribute, as defined below:

refers to both implicit and explicit subprograms.

## VASG-ISAC Recommendation for Future Revisions

Reword the first paragraph on page 5-2 as follows:

The attribute designator must denote an attribute. The entity name list identifies those entities, *both implicit and explicit*, that inherit the attribute, as defined below:

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

|                           |                                                            |
|---------------------------|------------------------------------------------------------|
| <b>Issue Report:</b>      | <b>19</b>                                                  |
| <b>Classification:</b>    | <b>Language Definition</b>                                 |
| <b>Summary:</b>           | <i>Meanings of names in association lists are unclear.</i> |
| <b>Related IRs:</b>       | 60                                                         |
| <b>Relevant Sections:</b> | 4.3.3.2, 10                                                |
| <b>Key Words/Phrases:</b> | association lists, select visibility                       |
| <b>Current Status:</b>    | VASG Approved                                              |
| <b>Date Submitted:</b>    | 10 February 1989                                           |
| <b>Date Last Revised:</b> | 27 June 1990                                               |

---

## Description of the Problem

$X(Y)$  is ambiguous as a formal if  $X$  is a visible (type-conversion) function and there is a formal array  $X$ .  $X$  is ambiguous as an actual if there is a local  $X$  and a normally visible  $X$ . It is unclear if  $X+4$  as an actual in a generic map is valid if  $X$  denotes a local generic. If this is valid, it is unclear if the association counts as the required association for  $X$ .

Note that select visibility is rather botched in VHDL. This notion was taken from Ada but adapted poorly. First, in VHDL, there is select visibility over wide regions of text (e.g., expressions) as opposed to in Ada where it is defined on an identifier basis. Second, in regions to which select visibility applies, normal direct visibility is also possible.

## Proposed Resolution

If  $X(Y)$  appears as a formal,  $X$  is a visible (type-conversion) function, and there is a formal  $X$ , it is assumed this is a reference to a subelement of the formal  $X$ . If  $X$  appears as an actual in a binding indication and there is a local  $X$  and a normally visible  $X$ , it is assumed this is a reference to the local  $X$ . Hence in both cases, “priority” is given to the select-visibility interpretation. If  $X+4$  appears as an actual in a binding indication,  $X$  cannot refer to a local. References to locals in such contexts are only considered valid if the actual is in the form of a name.

---

## VASG-ISAC Analysis and Rationale

The visibility problems in this area of the standard are broader than is suggested here. In particular, they are not confined to association lists since the standard does not say, in general, who “wins” when one declaration is visible by selection and another declaration is directly visible.

It seems to make most sense that if a declaration is visible by selection at a particular point, no declaration should be directly visible at that point. This seems to be the approach taken by Ada, although the wording in the Ada LRM (i.e., the paragraph beginning “Where it is not visibly by selection, ...” in Section 8.3 of the Ada LRM) appears somewhat confused.

What remains to be solved are the problems of select visibility in VHDL association lists. These problems are not present in Ada because Ada does not support subelement association. Furthermore, Ada does not have a need for select visibility on the right-hand side of a named association element.

Two approaches seem possible for the problems of select visibility in VHDL association lists. Consider, for example, a parameter association list association element  $A \Rightarrow B$ , where  $A$  and  $B$  represent sequences of symbols, and consider the need to make the formal parameters of the subprogram visible (at least somewhere) within  $A$ . One approach would make select visibility “applicable” for each identifier in  $A$ . This is a simple way to work with the problem, but it makes select visibility applicable for some identifiers unnecessarily, which can result in conflicts (e.g.,  $Y$  in  $X.Y(\dots) \Rightarrow B$ ) and which would have to be resolved with additional rules (e.g., “.” visibility has higher priority). The alternative approach is to say that select visibility is only applicable for the left identifier in  $A$ . An additional statement would be required with the second approach that the parameter to a type conversion function appearing in an association element is visible by selection.

The second of these approaches seems simpler and cleaner and is therefore the preferred solution that is detailed below.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

### Clarification 1:

The second paragraph following the list of 11 items in Section 10.3 leads with the sentence:

Finally, a declaration is said to be *directly visible* within a certain part of its immediate scope; this part extends to the end of the immediate scope of the declaration, but excludes places where the declaration is hidden as explained below.

This sentence should be read as though it were written:

Finally, a declaration is said to be *directly visible* within a certain part of its immediate scope; this part extends to the end of the immediate scope of the declaration, but excludes places where the declaration is hidden (as explained below) or where any declaration with the same identifier, operator symbol, or character literal is visible by selection.

### Clarification 2:

Items 7, 8, 9, 10, and 11 in the list of 11 items in Section 10.3 each contain the phrase

... at the place of the formal designator (before the compound delimiter  $\Rightarrow$ ) in a ...

These phrases should be read as though they were written

... at the place of the left identifier, operator symbol, or character literal in the formal part of a ...

### Clarification 3:

Items 8 and 9 in the list of 11 items in Section 10.3 each contain the phrase

... at the place of the actual designator (after the compound delimiter  $\Rightarrow$ ) in a ...

These phrases should be read as though they were written

... at the place of the left identifier, operator symbol, or character literal in the actual part of a ...

*Clarification 4:*

The list of 11 items in Section 10.3 should be read as though it contained the following two additional items:

12. For a formal parameter declaration of a given subprogram declaration, a local generic or local port declaration of a given component declaration, or a formal generic or formal port declaration of a given entity declaration: at the place of the left identifier, operator symbol, or character literal in the formal designator of a formal part that is in the form of a function call in a named association element of a corresponding subprogram call, component instantiation statement, or binding indication, respectively.
13. For a local generic or local port declaration of a given component declaration: at the place of the left identifier, operator symbol, or character literal in the actual designator of an actual part that is in the form of a function call in an association element of a corresponding binding indication.

## VASG-ISAC Recommendation for Future Revisions

It is not clear that this is the “best” long-term solution to these problems. These issues should be re-examined when the next revision of the standard is being prepared.

**NOTE:** There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987.

### Issue Report: 19

#### Formal Comments

There are still some problems in the resolution to this issue, especially in Clarifications 2 and 3. Consider the actuals “L+1” and “1+L.” They are treated differently!

The intent of the introduction of “left” was to treat the type conversion function reference differently from the formal or actual being type converted with respect to visibility. Perhaps the best approach is to split items 7–11 into multiple cases, one for nontype converted formals and actuals, and the other for type converted formals and actuals.

In any case, other elements of visibility need extensive work (e.g., see IR 41),\* so this can be corrected for the next revision of the standard.

**NOTE:** These formal comments are not part of the official interpretations of this document.

---

\*Unapproved at the time of publication.

## Issue Report:

Classification:

Summary:

Related IRs:

Relevant Sections:

Key Words/Phrases:

Current Status:

Date Submitted:

Date Last Revised:

## 20

### Language Definition

*Restrictions on use of deferred constants are overly restrictive.*

8, 59

2.2, 2.6, 4.3.1.1, 12.5

deferred constant, dynamic elaboration, full declaration, package, package body, subprogram body

VASG Approved

23 February 1989

9 October 1990

## Description of the Problem

Consider the following VHDL design units:

```
package P is
  constant C: Integer;      —a deferred constant
  —Place 1
end P;
```

```
package body P is
  —Place 2
```

```
  procedure Proc is
    —Place 3
  begin
    —Place 4
  end Proc;
```

```
  constant C : Integer := 0; —the full declaration of the deferred constant
end P;
```

The LRM states, in paragraph 5 of Section 2.6, that references to C prior to its corresponding full declaration are allowed only under restricted circumstances. Specifically, a name denoting the deferred constant can be used only “in the default expression for a local generic, local port, or formal parameter.” Thus, Places 1 through 4 in the above example are treated identically. Since declarations appearing in Places 1 and 2 (except for those mentioned in the LRM quotation) require the value of C when those declarations are elaborated, the restriction is proper for these places.

This restriction is unnecessary in Places 3 and 4. While the elaboration of the subprogram body will occur prior to the elaboration of the full declaration of C, the elaboration of the subprogram declarative part (Place 3) and the execution of its statement part (Place 4) occur only when the subprogram is invoked. Thus, the restriction on the use of C in these places is unnecessary.

## Proposed Resolution

Modify the language of paragraph 5 of Section 2.6 to allow unrestricted reference to deferred constants before the full declaration corresponding to a specified deferred constant when the reference occurs within a subprogram body. Suggested language for the first sentence of paragraph 5 of Section 2.6 is:

Within a package declaration that contains the declaration of a deferred constant, and within the body of that package, before the end of the corresponding full declaration, the use of a name that denotes the deferred constant is only allowed within a subprogram body, or in the default expression for a local generic, local port, or formal parameter.

---

## VASG-ISAC Analysis and Rationale

The problem described above is accurate. However, the suggested solution does not adequately treat a case such as:

```
package P is  
  constant C: Integer;      —a deferred constant  
end P;  
  
package body P is  
  
  function Func return Integer is  
    —Place 3  
  begin  
    —Place 4  
  end Proc;  
  
  constant K : Integer := Func;  
  
  constant C : Integer := 0; —the full declaration of the deferred constant  
end P;
```

For a situation such as this, a use of a deferred constant in Places 3 or 4 will be erroneous because function Func is invoked prior to the elaboration of the full declaration of the deferred constant. Rather than complicating the proposed rule to disallow this case while allowing the original example, nothing shall be done about this problem for now. It is true that the rules are technically overly restrictive, but it does not seem this problem arises often in practice. In fact, it seems users typically supply the full declaration of the deferred constant at the top of the package body; there is simply no advantage to burying the full declaration well into the package body.

## VASG-ISAC Interpretation for IEEE Std 1076-1987

VHDL users should follow the current rules regarding deferred constants since little or no hardship is imposed in practice.

## VASG-ISAC Recommendation for Future Revisions

If a simple rule could be found that would relax restrictions on the use of deferred constants and still be safe, this should be considered.

|                                                                                                              |
|--------------------------------------------------------------------------------------------------------------|
| NOTE: There is no certainty that these recommendations will become part of a revision to IEEE Std 1076-1987. |
|--------------------------------------------------------------------------------------------------------------|

|                      |                                                                    |
|----------------------|--------------------------------------------------------------------|
| <b>Issue Report:</b> | <b>21</b>                                                          |
| Classification:      | <b>Language Definition</b>                                         |
| Summary:             | <i>Nonlocally static universal expressions must be restricted.</i> |
| Related IRs:         | None                                                               |
| Relevant Sections:   | 7.5                                                                |
| Key Words/Phrases:   | locally static expressions, universal expressions                  |
| Current Status:      | VASG Approved                                                      |
| Date Submitted:      | 10 February 1989                                                   |
| Date Last Revised:   | 1 October 1990                                                     |

---

## Description of the Problem

It would be a significant burden to support universal arithmetic at elaboration time. The intention of Section 7.5 is to prevent this. However, “static” appears in this paragraph where “locally static” should appear.

## Proposed Resolution

We assume “static expression” and “nonstatic universal expression” in the last two paragraphs of Section 7.5 are replaced with “locally static expression” and “nonlocally static universal expression,” respectively.

---

## VASG-ISAC Analysis and Rationale

Section 7.5 of the standard, which states in part

... if a universal expression is a static expression, then the evaluation must be exact.

was adopted nearly intact from Section 4.10 of the Ada LRM. The only changes were to remove references to Ada concepts not appearing in VHDL, such as exceptions and objects in package SYSTEM.

Ada defines only one kind of static expression, the kind that is compile-time computable (see Section 4.9 of the Ada LRM). However, VHDL defines two: one that is analysis-time computable (“locally static expressions”) and another that is computable during model elaboration (“globally static expressions”)—see Section 7.4 of IEEE Std 1076-1987. VHDL further defines the term “static expression” to encompass both locally static expressions and globally static expressions.

Therefore, IEEE Std 1076-1987 requires exact evaluation of universal expressions during analysis and during model elaboration, while the Ada LRM requires exact evaluation of universal expressions only during compilation. This is the crux of the author’s issue.

The issue author wishes to make VHDL conform to Ada by substituting the term “locally static” for the occurrence of “static” in Paragraph 3 of Section 7.5 of IEEE Std 1076-1987, and by substituting “nonlocally static” for “nonstatic” in Paragraph 4 of the same section.

The author states that exact evaluation during model elaboration is “a significant burden.” This