

# IEEE Standard for VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification

Sponsor

**Design Automation Standards Committee  
of the  
IEEE Computer Society**

Approved December 12, 1995

**IEEE Standards Board**

**Abstract:** The VITAL (VHDL Initiative Towards ASIC Libraries) ASIC Modeling Specification is defined. It creates a methodology that promotes the development of highly accurate, efficient simulation models for ASIC (Application-Specific Integrated Circuit) components in VHDL.

**Keywords:** ASIC, computer, computer languages, constraints, delay calculation, HDL, modeling, SDF, timing, Verilog®, VHDL

---

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

Copyright © 1996 by the Institute of Electrical and Electronics Engineers, Inc.  
All rights reserved. Published 1996. Printed in the United States of America.

ISBN 1-55937-691-0

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.

**IEEE Standards** documents are developed within the Technical Committees of the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Board. Members of the committees serve voluntarily and without compensation. They are not necessarily members of the Institute. The standards developed within IEEE represent a consensus of the broad expertise on the subject within the Institute as well as those activities outside of IEEE that have expressed an interest in participating in the development of the standard.

Use of an IEEE Standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments.

**Interpretations:** Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of all concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason IEEE and the members of its technical committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments on standards and requests for interpretations should be addressed to:

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

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying all patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Authorization to photocopy portions of any individual standard for internal or personal use is granted by the Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; (508) 750-8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

# Introduction

[This introduction is not a part of IEEE Std 1076.4-1995, IEEE Standard for VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification.]

The objective of VITAL (VHDL Initiative Towards ASIC Libraries) was to accelerate the development of sign-off quality ASIC macrocell simulation libraries written in the VHSIC Hardware Description Language (VHDL) by leveraging existing methodologies of model development.

The VITAL effort germinated from ideas generated at the VHDL International Users' Forum in May 1992. Further discussions revealed that the biggest impediment to VHDL design was the lack of ASIC libraries, and that the biggest impediment to ASIC library development was the lack of a uniform, efficient method for handling timing in VHDL. Since this problem had already been solved for other languages, it was clear that a solution in VHDL was possible and that an effective way to arrive at this solution was to leverage existing technology. Leveraging existing tools and environments serves as a catalyst for the rapid deployment of ASIC libraries.

VITAL obtains its leverage from

- a) The Standard Delay Format (SDF), in addition to
- b) Certain contributed elements of the Std\_Timing package provided by William Billowitch and specialized timing and behavioral techniques provided by Ray Ryan, and
- c) The existence of numerous ASIC libraries and tools developed using SDF timing implementations

VITAL used many ideas about the primitives and timing models described in the Verilog<sup>®</sup> language. In particular, Verilog's support for representing truth/state tables and its mechanism for performing pin-to-pin delay selection were found to be highly useful.

The VITAL organization was an informal consortium of interested companies in the electronics industry who shared the goals of accelerating the availability of VHDL ASIC libraries. They represented the three components of the ASIC design triangle: ASIC vendors, EDA vendors, and end users of ASIC components. More than 60 such companies worldwide joined the consortium. This group was led by a steering committee, which was responsible for the development of the VITAL technical specifications as well as the promotion and dissemination of information about this work. The members of the steering committee were: Steven Schulz, William Billowitch, Ray Ryan, Oz Levia, Victor Berman, Victor Martin, Ravi Kumar, Sanjay Nayak, Tom Senna, and Herman Van Beek.

The VITAL steering committee transferred this work to the IEEE P1076.4 Working Group for consideration as the basis of a standard for ASIC modeling in VHDL. This standard is the result of the efforts of the working group in refining that baseline document.

The IEEE P1076.4 Working Group has a membership of over 300 interested people who have made significant contributions to this work through their participation in technical meetings; their review of technical data both in print and through electronic media; and their votes, which guided and finally approved the content of the draft standard.

The technical direction of the working group as well as the day-to-day activities of issue analysis and drafting of proposed wordings for the standard were the responsibility of the IEEE P1076.4 TAG (Technical Action Group). This group consisted of Ravi Kumar, Sanjay Nayak, Dennis Brophy, Ray Ryan, John Busco, Tim Ayres, Bill Paulsen, and Kathy McKinley, and the group was chaired by Oz Levia. Without the dedication and hard work of this group, it would not have been possible to complete this work.

This standard is the result of numerous discussions with ASIC vendors, CAE tool vendors, and ASIC designers

to determine the requirements for effective design and fabrication of ASICs using VHDL. The highest priority issues identified by this group were

- Timing accuracy
- Model maintainability
- Simulation performance

Some basic guiding principles followed during the entire specification development process were

- To describe all functionality and timing semantics of the model entirely within the VHDL model and the associated VITAL packages.
- To provide a set of modeling rules (Level 1) that constrain the use of VHDL to a point that is amenable for simulator optimizations, and at the same time to provide enough flexibility to support most existing modeling scenarios.
- To have all timing calculations (load dependent or environmentally dependent) performed outside of the VITAL model. The VITAL model would get these timing values solely as actual values to the generic parameter list of the model or via SDF direct import.

The following persons were members of the IEEE P1076.4 Working Group:

**Victor Berman, *Chair***

**Kathy McKinley, *Technical Editor***

Suresh Agarwal  
Dave Agnew  
Dave Allenbaugh  
Brien Anderson  
Cliff Anderson  
Jan Anderson  
Robert E. Anderson  
Ronen Arad  
Libby Aston  
Naveena N. Aswadhati  
Jeffrey M. Aubert  
Larry M. Augustin  
Bulent Ay  
Tim Ayres  
Stephen A. Bailey  
Mikhail A. Baklashov  
Ekambaram Balaji  
Bruce Bandali  
Sudarshan Banerjee  
Peter Barck  
Daniel Barclay  
Karen Bartleson  
Mark Basten  
Michael A. Beaver  
David Belz  
Leon Benders  
Ad J. W. M. ten Berg  
Jean-Michel Berge  
Werner Bergmann  
David Bernstein

Christian Berthet  
Jayaram Bhasker  
Scott Bilik  
William D. Billowitch  
John Biro  
Dave Bishop  
Alan Blair  
Robert Bloor  
Michael Bohm  
Frederique Bouchard  
Bruce Bourbon  
Elaine Boyd  
Jean Pierre Braunt  
Christopher Brock  
Dennis Brophy  
John A. Busco  
Jean-Paul Caisso  
J. Scott Calhoun  
Brian Caslis  
Shir-Shen Chang  
Thomas Chao  
Clive Charlwood  
Jean Marc Chateau  
Glenn J. Childers  
Nitin Chowdhary  
Debashis Roy Chowdhury  
Ron Christopher  
Carl Cleaver  
David Coelho  
John Colley

Tedd Corman  
Joe Costello  
Terry Coston  
Ronnie Craig  
Ronan Cullen  
Hal Daseking  
Jose De Castro  
Aart deGeus  
Jean Desuche  
Baruch Deutsch  
Allen Dewey  
Matt Dodd  
Kevin Donnelly  
Mike Duclos  
Douglas D. Dunlop  
Hisakazu Edamatsu  
Steve Elliott  
Stephan Eriksson  
David Evans  
Edward N. Evans  
John A. Evans  
William Fazakerly  
Francine Ferguson  
Len Finegold  
Geoffrey Frank  
Roberta E. Fulton  
Sanjay Goswami  
Ravender Goyal  
Brian Griffin  
Gary Griffin

Bruce Gruett  
Andrew Guyler  
Werner Hack  
Klaus ten Hagen  
Stuart Hamilton  
James P. Hanna  
Alain Hanover  
Susan Hardenbrook  
David Hardman  
Randolph E. Harr  
Amirhooshang Hashemi  
Greg Haynes  
Carl E. Hein  
Mitch Heins  
Shankar Hemmady  
Moises Hernandez  
Elhanan Herzog  
Kiyohiro Higashijima  
John Hillawi  
Fred Hinchcliffe  
John Hines  
Helmut Hissen  
Will Hobbs  
Andreas Hohl  
Yu-I Hsieh  
Jason Hsiung  
Anling Hsu  
Matt Hsu  
Pong Hsu  
May Huang  
David Hubbard  
Christophe Hui-Bon-Hoa  
Erik Huyskens  
Kazuhiko Iijima  
Stephen A. Ives  
Jacob Jacobson  
Antonie C. S. de Jager  
Mahendra Jain  
Sunil K. Jain  
Colin Jitlal  
Don Johansen  
Howard Johnson  
Harvey Jones  
Mark Jones  
Masaru Kakimoto

Ken Kappeler  
Jake Karrfalt  
Eugene Ko  
Satoshi Kojima  
Rama Kowsalya  
Stanley J. Krolikoski  
Paul Krueger  
Krishna Kumar  
Ravi Kumar  
C. Lakshmikantam  
Sylvie Lasserre  
Marc Laurent  
Manu Lauria  
Oz Levia  
Serge Maginot  
Arthur Magnan  
Sanjay Malpani  
John Mancini  
Maq Mannan  
F. Erich Marschner  
Victor M. Martin  
Larry Melling  
Paul J. Menchini  
Alex Miczo  
Dwight Miller  
Nikhil Modi  
Kent Moffat  
Sidhartha Mohanty  
Gabe Moretti  
Rick Munden  
Jayant L. Nagda  
Zain Navabi  
Sanjay Nayak  
Gordon L. Newell  
John O'Brien  
William R. Paulsen  
Zamir Paz  
Venu Pemmaraju  
Gabriele Pulini  
Viswanathan Ramakrishnan  
Paul Ramondetta  
Prakash Reddy  
Praveen Reddy  
Eugen Roehm  
Adam Rosenberg

Arnob Roy  
Susan Runowicz-Smith  
Ray Ryan  
Yvonne Ryan  
Hideki Sagayama  
Kyohei Sakajiri  
Manohar Sambandam  
Johan Sandstrom  
Marc-Alain Santerre  
Souvanik Sarkar  
Paul Sathya  
Sharada Satrasala  
Larry F. Saunders  
Quentin Schmierer  
Joel Schoen  
Steven E. Schulz  
David Sellers  
Tom Senna  
Moe Shahdad  
Steve Sherman  
Venk Shukla  
Ken Simone  
Peter Sinander  
Rajvinder Singh  
John Sissler  
Joseph Skudlarek  
Balachandran Sreekandath  
John Stickley  
Eugena Talvola  
Tina Tran  
Cary Ussery  
Alain Vachoux  
Vijay Vaidyanathan  
Tom VandenBerge  
Eugenio Villar  
Walter Vines  
Jeffery Vo  
Russ Vreeland  
Ron Werner  
Paul Williams  
John C. Willis  
Jim Wilmore  
Scott Winick  
Alex Zamfirescu

The following persons were on the balloting committee:

William J. Abboud  
Mostapha Aboulhamid  
Robert E. Anderson  
Stephen A. Bailey  
Pete Bakowski  
Jean-Michel Berge  
Victor Berman  
Jayaram Bhasker  
William D. Billowitch  
Dennis B. Brophy  
John A. Busco  
J. Scott Calhoun  
Raul Camposano  
Harold W. Carter  
Shir-Shen Chang  
Thomas Chao  
Chin-Fu Chen  
Mojy C. Chian  
Pradeep Chilka  
David Coelho  
John Colley  
Alan Coppola  
Tedd Corman  
W. Terry Coston  
Joanne DeGroat  
Antonie C. deJager  
Allen Dewey  
Michael A. Dukes  
Douglas D. Dunlop  
Ted Elkind  
Edward N. Evans  
William Fazakerly  
Peter Flake  
Jacques P. Flandrois  
Rita A. Glover

Rich Goldman  
Brian Griffin  
Richard Grisel  
Steve Grout  
Andrew Guylar  
William A. Hanna  
James P. Hanna  
Randolph E. Harr  
John Hillawi  
Robert G. Hillman  
Yu-I Hsieh  
Yee-Wing Hsieh  
May Huang  
Christophe Hui-Bon-Hoa  
Sylvie Hurat  
Masaharu Imai  
Ann Irza  
Mitsuaki Ishikawa  
Stephen A. Ives  
Navneet Kumar Jain  
Choon B. Kim  
Stanley J. Krolikoski  
Jean Lebrun  
Oz Levia  
Shirley Lu  
Rajeev Madhavan  
Serge Maginot  
F. Erich Marschner  
Victor M. Martin  
Paul J. Menchini  
Gerald T. Michael  
John T. Montague  
Gabe Moretti  
Jayant L. Nagda

Sanjay Nayak  
Kevin O'Brien  
Curtis Parks  
William R. Paulsen  
Mauro Pipponzi  
Gary S. Porter  
Adam Postula  
Jan Pukite  
Rami Rahim  
Hemant G. Rotithor  
Larry F. Saunders  
Quentin Schmierer  
Steven E. Schulz  
Francesco Sforza  
Ravi Shankar  
Micahel D. Sharp  
Raj Singh  
Djahida Smati  
Geoffrey John Smith  
J. Dennis Soderberg  
Alec G. Stanculescu  
Balsha R. Stanisic  
Michael F. Sullivan  
Charles Swart  
Peter Ting  
Cary Ussery  
E. Vandris  
Ranganadha R. Vemuri  
Venkat V. Venkataraman  
Eugenio Villar  
Ronald Waxman  
Ron Werner  
Alan Whittaker  
John C. Willis  
Mark Zwolinski

When the IEEE Standards Board approved this standard on December 12, 1995, it had the following membership:

**E. G. “Al” Kiener, *Chair***

**Donald C. Loughry, *Vice Chair***

**Andrew G. Salem, *Secretary***

Gilles A. Baril  
Clyde R. Camp  
Joseph A. Cannatelli  
Stephen L. Diamond  
Harold E. Epstein  
Donald C. Fleckenstein  
Jay Forster\*  
Donald N. Heirman  
Richard J. Holleman

Jim Isaak  
Ben C. Johnson  
Sonny Kasturi  
Lorraine C. Kevra  
Ivor N. Knight  
Joseph L. Koepfinger\*  
D. N. “Jim” Logothetis  
L. Bruce McClung

Marco W. Migliaro  
Mary Lou Padgett  
John W. Pope  
Arthur K. Reilly  
Gary S. Robinson  
Ingo Rutsch  
Chee Kiow Tan  
Leonard L. Tripp  
Howard L. Wolfman

\*Member Emeritus

Also included are the following nonvoting IEEE Standards Board liaisons:

Satish K. Aggarwal  
Steve Sharkey  
Robert E. Hebner  
Chester C. Taylor

Mary Lynne Nielsen  
*IEEE Standards Project Editor*

Verilog is a registered trademark of Cadence Design Systems, Inc.

# Contents

| CLAUSE                                                                   | PAGE |
|--------------------------------------------------------------------------|------|
| 1. Overview .....                                                        | 1    |
| 1.1 Intent and scope of this standard .....                              | 1    |
| 1.2 Structure and terminology of this standard .....                     | 1    |
| 1.3 Syntactic description .....                                          | 1    |
| 1.4 Semantic description .....                                           | 3    |
| 1.5 Front matter, examples, figures, notes, and annexes.....             | 3    |
| 2. References .....                                                      | 4    |
| 3. Basic elements of the VITAL ASIC modeling specification .....         | 5    |
| 3.1 VITAL modeling levels and compliance .....                           | 5    |
| 3.2 VITAL standard packages.....                                         | 6    |
| 3.3 VITAL specification for timing data insertion.....                   | 6    |
| 4. The VITAL Level 0 specification .....                                 | 8    |
| 4.1 The VITAL_Level0 attribute .....                                     | 8    |
| 4.2 General usage rules .....                                            | 8    |
| 4.2.1 Standard VHDL usage.....                                           | 8    |
| 4.2.2 Organization of VITAL-compliant descriptions .....                 | 9    |
| 4.3 The VITAL Level 0 entity interface .....                             | 9    |
| 4.3.1 Ports .....                                                        | 10   |
| 4.3.2 Generics .....                                                     | 10   |
| 4.3.2.1 Timing generics .....                                            | 10   |
| 4.3.2.2 Control generics.....                                            | 17   |
| 4.4 The VITAL Level 0 architecture body.....                             | 17   |
| 4.4.1 Timing generic usage.....                                          | 18   |
| 5. Backannotation.....                                                   | 19   |
| 5.1 Backannotation methods .....                                         | 19   |
| 5.1.1 Direct SDF import .....                                            | 19   |
| 5.1.2 The SDF annotator.....                                             | 19   |
| 5.2 The VITAL SDF map .....                                              | 20   |
| 5.2.1 Delay file .....                                                   | 20   |
| 5.2.2 Header section .....                                               | 20   |
| 5.2.3 CELL entry .....                                                   | 21   |
| 5.2.4 INSTANCE and CELLTYPE entries .....                                | 21   |
| 5.2.5 Timing specifications .....                                        | 22   |
| 5.2.6 Data value mapping .....                                           | 23   |
| 5.2.7 Mapping to timing generics.....                                    | 23   |
| 5.2.7.1 DELAY entry.....                                                 | 23   |
| 5.2.7.2 TIMINGCHECK entry .....                                          | 26   |
| 5.2.7.3 Mapping of SDF constructs to general VHDL lexical elements ..... | 28   |
| 6. The VITAL Level 1 specification .....                                 | 33   |

| CLAUSE                                                   | PAGE |
|----------------------------------------------------------|------|
| 6.1 The VITAL_Level1 attribute .....                     | 33   |
| 6.2 The VITAL Level 1 architecture body.....             | 33   |
| 6.3 The VITAL Level 1 architecture declarative part..... | 34   |
| 6.3.1 VITAL internal signals.....                        | 34   |
| 6.4 The VITAL Level 1 architecture statement part.....   | 34   |
| 6.4.1 Wire delay block statement .....                   | 35   |
| 6.4.2 Negative constraint block statement.....           | 37   |
| 6.4.3 VITAL process statement.....                       | 37   |
| 6.4.3.1 VITAL process declarative part.....              | 38   |
| 6.4.3.2 VITAL process statement part.....                | 39   |
| 6.4.4 VITAL primitive concurrent procedure call.....     | 41   |
| 7. Predefined primitives and tables .....                | 43   |
| 7.1 VITAL logic primitives .....                         | 43   |
| 7.1.1 Logic primitive functions .....                    | 43   |
| 7.1.2 Logic primitive procedures.....                    | 44   |
| 7.1.3 Establishing output strengths.....                 | 45   |
| 7.2 VitalResolve.....                                    | 45   |
| 7.3 VITAL table primitives.....                          | 45   |
| 7.3.1 VITAL table symbols .....                          | 45   |
| 7.3.2 Table symbol matching .....                        | 46   |
| 7.3.3 TruthTable primitive.....                          | 47   |
| 7.3.3.1 Truth table construction .....                   | 48   |
| 7.3.3.2 TruthTable algorithm.....                        | 49   |
| 7.3.4 StateTable primitive .....                         | 49   |
| 7.3.4.1 State table construction.....                    | 49   |
| 7.3.4.2 StateTable algorithm.....                        | 50   |
| 8. Timing constraints.....                               | 51   |
| 8.1 Timing check procedures .....                        | 51   |
| 8.1.1 VitalSetupHoldCheck.....                           | 51   |
| 8.1.2 VitalPeriodPulseCheck.....                         | 52   |
| 8.1.3 VitalRecoveryRemovalCheck .....                    | 52   |
| 8.2 Modeling negative timing constraints.....            | 53   |
| 8.2.1 Requirements on the VHDL description .....         | 54   |
| 8.2.2 Negative constraint calculation phase .....        | 55   |
| 8.2.2.1 Calculation of internal clock delays .....       | 55   |
| 8.2.2.2 Calculation of internal signal delays.....       | 56   |
| 8.2.2.3 Calculation of biased propagation delays .....   | 56   |
| 8.2.2.4 Adjustment of propagation delay values .....     | 56   |
| 8.2.2.5 Adjustment of timing check generics .....        | 57   |
| 9. Delay selection .....                                 | 59   |
| 9.1 VITAL delay types and subtypes.....                  | 59   |
| 9.2 Transition-dependent delay selection.....            | 60   |
| 9.3 Glitch handling.....                                 | 60   |
| 9.4 Path delay procedures .....                          | 61   |
| 9.4.1 VitalPathDelay andVitalPathDelay01 .....           | 62   |
| 9.4.2 VitalPathDelay01Z .....                            | 62   |



| CLAUSE                                       | PAGE |
|----------------------------------------------|------|
| 9.5 Delay selection in VITAL primitives..... | 63   |
| 9.6 VitalExtendToFillDelay .....             | 63   |
| 10. The VITAL standard packages .....        | 65   |
| ANNEX                                        |      |
| Annex A Syntax summary (informative) .....   | 66   |
| Annex B Glossary (informative) .....         | 70   |
| Annex C Bibliography (informative) .....     | 71   |
| Index .....                                  | 72   |

# IEEE Standard for VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification

## 1. Overview

This clause describes the purpose and organization of this standard.

### 1.1 Intent and scope of this standard

The intent of this standard is to define accurately the standard VITAL ASIC modeling specification. The primary audience for this document are the implementors of tools supporting the standard and ASIC modelers.

### 1.2 Structure and terminology of this standard

This standard is organized into clauses, each of which focuses on some particular area of the definition of the specification. Each page of the formal definition contains ruler-style line numbers in the left margin. Within each clause, individual constructs or concepts are discussed in each subclause.

Each subclause describing a specific construct or concept begins with an introductory paragraph. If applicable, the syntax of the construct is then described using one or more grammatical productions. A set of paragraphs describing in narrative form the information and rules related to the construct or concept then follows. Finally, each subclause may end with examples, figures, and notes.

### 1.3 Syntactic description

The form of a VITAL-compliant VHDL description is described by means of a context-free syntax, using a simple variant of the Backus-Naur form; in particular:

- a) Lowercased words, some containing embedded underlines, are used to denote syntactic categories, for example:

VITAL\_process\_statement

Whenever the name of a syntactic category is used, apart from the syntax rules themselves, spaces take the place of underlines (thus, "VITAL process statement" would appear in the narrative description when referring to the above syntactic category).

- b) Boldface words are used to denote reserved words, for example:

**process**

Reserved words shall be used only in those places indicated by the syntax.

- c) A *production* consists of a *left-hand side*, the symbol “::=” (which is read as “can be replaced by”), and a *right-hand side*. The left-hand side of a production is always a syntactic category; the right-hand side is a replacement rule.

The meaning of a production is a textual-replacement rule: any occurrence of the left-hand side may be replaced by an instance of the right-hand side.

- d) A vertical bar separates alternative items on the right-hand side of a production unless it occurs immediately after an opening brace, in which case it stands for itself.
- e) Square brackets enclose optional items on the right-hand side of a production.
- f) Braces enclose a repeated item or items on the right-hand side of a production. The items may appear zero or more times; the repetitions occur from left to right as with an equivalent left-recursive rule.
- g) If the name of any syntactic category starts with an italicized part, it is equivalent to the category name without the italicized part. The italicized part is intended to convey some semantic information. For example, *unrestricted\_variable\_name* is syntactically equivalent to name alone.
- h) The term `simple_name` is used for any occurrence of an identifier that already denotes some declared entity.
- i) A syntactic category for which no replacement rule is specified is assumed to correspond to the VHDL syntactic category of the same name. In this case the appropriate replacement rule can be found in IEEE Std 1076-1987.<sup>1</sup>
- j) A syntactic category beginning with the unitalicized prefix “VITAL\_” represents a subset of a VHDL syntactic category.
- k) A word shown in all uppercase letters can represent a reserved word in VHDL.
- l) SDF constructs are represented in the following manner:
- 1) An italicized lowercase identifier represents an SDF syntax construct.
  - 2) The definition of a syntax construct is indicated by the symbol ::=, and alternative definitions are separated by the symbol ||=.
  - 3) Keywords appear in boldface capital letters.
  - 4) Uppercase identifiers represent variable symbols.
  - 5) The form “item?” represents an optional item.

<sup>1</sup>Information about references can be found in clause 2.

- 6) The form “item\*” represents zero or more occurrences of the item.
- 7) The form “item+” indicates one or more occurrences of the item.

## 1.4 Semantic description

The meaning of a particular construct or concept and any related restrictions are described with a set of narrative rules immediately following any syntactic productions in the subclause. In these rules, an italicized term indicates the definition of that term, and an identifier appearing in **Helvetica** font refers to a definition in one of the VHDL or VITAL standard packages or in a VHDL model description. An identifier beginning with the prefix “VITAL” corresponds to a definition in a VITAL standard package.

Use of the words “is” or “shall” in such a narrative indicates mandatory weight. A noncompliant practice may be described as *erroneous* or as an *error*. These terms are used in these semantic descriptions with the following meaning:

*erroneous*—the condition described represents a noncompliant modeling practice; however, implementations are not required to detect and report this condition. Conditions are deemed erroneous only when it is either very difficult or impossible in general to detect the condition during the processing of a model.

*error*—the condition described represents a noncompliant modeling practice; implementations are required to detect the condition and report an error to the user of the tool.

## 1.5 Front matter, examples, figures, notes, and annexes

Prior to this clause are several pieces of introductory material; following the final clause are some annexes and an index. The front matter, annexes, and index serve to orient and otherwise aid the user of this standard but are not part of the definition of this standard.

Some subclauses of this standard contain examples, figures, and notes; with the exception of figures, these parts always appear at the end of a subclause. Examples are meant to illustrate the possible forms of the construct described. Figures are meant to illustrate the relationship between various constructs or concepts. Notes are meant to emphasize consequences of the rules described in the clause or elsewhere. In order to distinguish notes from the other narrative portions of the definition, notes are set as enumerated paragraphs in a font smaller than the rest of the text. Examples, figures, and notes are not part of the definition of the specification.

## 2. References

This standard shall be used in conjunction with the following publications. Bibliographic references may be found in annex C. Citations of the form “[B1]” refer to items listed in annex C, not to items listed in this clause.

IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual.<sup>2</sup>

IEEE Std 1164-1993, IEEE Standard Multivalued Logic System for VHDL Model Interoperability (Std\_logic\_1164).<sup>3</sup>

Standard Delay Format Specification, Version 2.1.<sup>4</sup>

---

<sup>2</sup>IEEE Std 1076-1987 has been superseded by IEEE Std 1076-1993 (see Annex C for bibliographic information on the latest version of this standard). IEEE Std 1076-1987 is no longer in print; however, it is available archivally from Global Engineering, 15 Inverness Way East, Englewood, CO 80112-5704, USA.

<sup>3</sup>IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA.

<sup>4</sup>OVI publications are available from Open Verilog International (OVI), 15466 Los Gatos Blvd, Suite 109-071, Los Gatos, CA 95032.

### 3. Basic elements of the VITAL ASIC modeling specification

This standard defines a modeling style for the purpose of facilitating the development and acceleration of sign-off quality ASIC macrocell simulation libraries written in VHDL.

This standard is an application of the VHSIC Hardware Description Language (VHDL), described in IEEE Std 1076-1987 and IEEE Std 1076-1993 [B1]. This standard uses the term *VHDL* to refer to the VHSIC Hardware Description Language. In no case shall the modeling rules introduced by this standard be in conflict with IEEE Std 1076-1987.

This standard relies on the IEEE Standard Multivalued Logic System for VHDL Model Interoperability (Std\_Logic\_1164), described in IEEE Std 1164-1993, for its basic logic representation. Throughout this standard, the term *standard logic* refers to the Std\_Logic\_1164 package or to an item declared in the Std\_Logic\_1164 package.

This standard relies on the Standard Delay Format (SDF), as defined by the OVI Standard Delay Format Specification Version 2.1, as a standard external timing data representation. Throughout this standard, the term *SDF* refers to this particular version of the delay format.

This standard consists of three basic elements: the formal definition of a VITAL-compliant VHDL model, a set of VHDL packages for providing standard timing support and standard functionality support, and a semantic specification describing a standard mechanism for insertion of timing data into a VHDL model.

#### 3.1 VITAL modeling levels and compliance

A VITAL ASIC cell is represented by a VHDL design entity. This standard defines the characteristics of a VITAL design entity in terms of the VHDL descriptions of the entity and architecture, and in terms of the associated model which is the result of the elaboration of those VHDL descriptions.

This standard defines two modeling levels; these levels are called *VITAL Level 0* and *VITAL Level 1*. Each modeling level is defined by a set of modeling rules. The VITAL Level 0 specification forms a proper subset of the VITAL Level 1 specification.

A model is said to adhere to the rules in a particular specification only if both the model and its VHDL description satisfy all of the requirements of the specification. Furthermore, if such a model makes use of an item described in a configuration declaration or a package other than a VHDL or VITAL standard package, then the external item shall satisfy the requirements of the specification, as though the item appeared in the VHDL description of the design entity itself.

The VITAL Level 0 specification defines a set of standard modeling rules that facilitate the portability and interoperability of ASIC models, including the specification of timing information. A model that adheres to the rules in the VITAL Level 0 specification is said to be a *VITAL Level 0 model*. The VITAL Level 0 modeling specification is described in clause 4.

The VITAL Level 1 specification defines a usage model for constructing complete cell models in a manner that

facilitates optimization of the execution of the models. A model that adheres to the rules in both the VITAL Level 0 model interface specification and the VITAL Level 1 model architecture specification is said to be a *VITAL Level 1 model*. The VITAL Level 1 modeling rules are defined in clause 6.

A model that is a VITAL Level 0 model or a VITAL Level 1 model is said to be *VITAL compliant*. A VITAL-compliant model description contains an attribute specification representing the highest level of compliance intended by the enclosing entity or architecture. Descriptions of these attribute specifications may be found in 4.1 and 6.1.

#### NOTES

1—A VITAL Level 1 model is by definition a VITAL Level 0 model as well (but not vice versa).

2—The rules outlined in the VITAL Level 0 and VITAL Level 1 specifications apply to model descriptions, not to the VITAL standard packages themselves.

3—A VITAL-compliant tool is assumed to enforce the definition of all applicable rules in accordance with the definitions of the terms IS, SHALL, ERROR, and ERRONEOUS. In addition, a compliant tool is expected to accept and execute correctly a VITAL-compliant model, and to identify and reject models that are not compliant. A VITAL-compliant tool is also expected to support fully the processes described in the specification, including SDF backannotation and negative time sequential constraint transformation.

### 3.2 VITAL standard packages

This standard defines two standard packages for use in specifying the timing and functionality of a model: *VITAL\_Timing* and *VITAL\_Primitives*. The text of these packages may be found in clause 10.

The *VITAL\_Timing* package defines data types and subprograms to support development of macrocell timing models. Included in this package are routines for delay selection, output scheduling, and timing violation checking and reporting.

The *VITAL\_Primitives* package defines a set of commonly used combinatorial primitives and general-purpose truth and state tables. The primitives are provided in both function and concurrent procedure form to support both behavioral and structural modeling styles.

### 3.3 VITAL specification for timing data insertion

This standard defines certain semantics that are assumed by a VITAL-compliant model and shall be implemented by a tool processing or simulating VITAL-compliant models that rely on these semantics. These semantics concern the specification and processing of timing data in a VHDL model. They cover SDF mapping, backannotation, and negative constraint processing.

The timing data for a VITAL-compliant model may be specified in Standard Delay Format (SDF). The VITAL SDF map is a mapping specification that defines the translation between SDF constructs and the corresponding generics in VITAL-compliant models. The mapping specification may be used by tools to insert timing information into a VHDL model, either by generating an appropriate configuration declaration or by performing backannotation through direct SDF import. The VITAL SDF map is defined in 5.2.

This standard introduces two new simulation phases for designs using VITAL models: the backannotation phase and the negative constraint calculation phase. These phases occur after VHDL elaboration but before initialization.

The backannotation specification defines a backannotation phase of simulation and a mechanism for directly

1 annotating generics with appropriate timing values from SDF (see 5.1.1). The specification also defines the correct  
state of the timing generics of a model at the end of the backannotation phase.

5 The negative constraint calculation specification describes a methodology for modeling negative timing  
constraints in VHDL (see 8.2). It defines a negative constraint calculation phase of simulation and an algorithm  
for computing and adjusting signal delays, which together transform the negative delays into nonnegatives for the  
purpose of simulation.

10

15

20

25

30

35

40

45

50

## 4. The VITAL Level 0 specification

The VITAL Level 0 specification is a set of modeling rules that promotes the portability and interoperability of model descriptions by outlining general standards for VHDL language usage, restricting the form and semantic content of VITAL Level 0 design entity descriptions, and standardizing the specification and processing of timing information. General VITAL Level 0 modeling rules are defined in this clause, and those relating to the modeling of negative timing constraints are defined in 8.2.1.

### 4.1 The VITAL\_Level0 attribute

A VITAL Level 0 entity or architecture is identified by its decoration with the VITAL\_Level0 attribute, which indicates an intention to adhere to the VITAL Level 0 specification.

VITAL\_Level0\_attribute\_specification ::= attribute\_specification

A VITAL Level 0 entity or architecture shall contain a specification of the VITAL\_Level0 attribute corresponding to the declaration of that attribute in the package VITAL\_Timing. The entity specification of the decorating attribute specification shall be such that the enclosing entity or architecture inherits the VITAL\_Level0 attribute. The expression in the VITAL\_Level0 attribute specification shall be the Boolean literal True.

NOTE—Because the required attribute specification representing VITAL compliance indicates the highest level of compliance (see 3.1), a VITAL Level 1 architecture, which is also by definition a VITAL Level 0 architecture, contains a VITAL\_Level1 attribute specification (see 6.1) rather than a VITAL\_Level0 attribute specification. The above rules apply to architectures that are only VITAL Level 0.

*Example:*

**attribute VITAL\_Level0 of VitalCompliantEntity : entity is True;**

### 4.2 General usage rules

A VITAL Level 0 model shall adhere to general usage rules that address portability and interoperability.

Rules that reference an item declared in a VHDL standard package, a VITAL standard package, or the Std\_Logic\_1164 package require the use of that particular item. A model description shall not use VHDL scope or visibility rules to declare or use an alternative item with the same name in the place of the item declared in one of these packages.

#### 4.2.1 Standard VHDL usage

A VITAL Level 0 model is restricted to the use of IEEE Std 1076-1987 features that are portable, as defined by IEEE Std 1076/INT-1991 [B2]. Use of foreign architecture bodies or package bodies is prohibited. In addition, a VITAL-compliant model shall not use any features of IEEE 1076-1987 that have been removed or modified in the language revision described in IEEE Std 1076-1993 [B1], nor shall it use any keywords or features introduced in IEEE Std 1076-1993 [B1].

It is erroneous for a VITAL model to make use of vendor-supplied attributes or other informative VHDL constructs, such as meta-comments or directives, in a manner that affects the function or timing characteristics of the model.

#### 4.2.2 Organization of VITAL-compliant descriptions

The VHDL design entity representing a VITAL ASIC cell is described by a pair of design units that reside in one or more VHDL design files. This standard imposes no special requirements on the placement of VITAL-compliant descriptions within design files, which may contain a mixture of compliant and noncompliant descriptions.

```
VITAL_design_file ::=
    VITAL_design_unit { VITAL_design_unit }

VITAL_design_unit ::=
    context_clause library_unit
| context_clause VITAL_library_unit

VITAL_library_unit ::=
    VITAL_Level_0_entity_declaration
| VITAL_Level_0_architecture_body
| VITAL_Level_1_architecture_body
```

#### 4.3 The VITAL Level 0 entity interface

A VITAL Level 0 entity declaration defines an interface between a VITAL-compliant model and its environment.

```
VITAL_Level_0_entity_declaration ::=
    entity identifier is
        VITAL_entity_header
        VITAL_entity_declarative_part
    end [ entity_simple_name ] ;

VITAL_entity_header ::=
    [ VITAL_entity_generic_clause ]
    [ VITAL_entity_port_clause ]

VITAL_entity_generic_clause ::=
    generic ( VITAL_entity_interface_list ) ;

VITAL_entity_port_clause ::=
    port ( VITAL_entity_interface_list ) ;

VITAL_entity_interface_list ::=
    VITAL_entity_interface_declaration { ; VITAL_entity_interface_declaration }

VITAL_entity_interface_declaration ::=
    interface_constant_declaration
    | VITAL_timing_generic_declaration
    | VITAL_control_generic_declaration
    | VITAL_entity_port_declaration

VITAL_entity_declarative_part ::= VITAL_Level0_attribute_specification
```

The form of this interface strictly limits the use of declarations and statements. The only form of declaration allowed in the entity declarative part is the specification of the `VITAL_Level0` attribute. No statements are allowed in the entity statement part.

### 4.3.1 Ports

Certain restrictions apply to the declaration of ports in a VITAL-compliant entity interface.

```
VITAL_entity_port_declaration ::=
    [ signal ] identifier_list : [ mode ] type_mark [ index_constraint ] [ := static_expression ] ;
```

The identifiers in an entity port declaration shall not contain underscore characters.

A port that is declared in an entity port declaration shall not be of mode `LINKAGE`.

The type mark in an entity port declaration shall denote a type or subtype that is declared in package `Std_Logic_1164`. The type mark in the declaration of a scalar port shall denote the subtype `Std_Ulogic` or a subtype of `Std_Ulogic`. The type mark in the declaration of an array port shall denote the type `Std_Logic_Vector`.

NOTE—The syntactic restrictions on the declaration of a port in a VITAL Level 0 entity are such that the port cannot be a guarded signal. Furthermore, the declaration cannot impose a range constraint on the port, nor can it alter the resolution of the port from that defined in the standard logic package.

### 4.3.2 Generics

The generics declared in a VITAL Level 0 entity generic clause may be timing generics, control generics, or other generic objects. Timing generics and control generics serve a special purpose in a VITAL-compliant model; specific rules govern their declaration and use. Other generics may be defined to control functionality; such generics are not subject to the restrictions imposed on timing or control generics.

#### 4.3.2.1 Timing generics

This standard defines a number of *timing generics*, which represent specific kinds of timing information. Each kind of timing generic is classified as either a *backannotation timing generic* or a *negative constraint timing generic*, depending on whether the value of the generic is set during the backannotation phase of simulation or the negative constraint calculation phase of simulation. Rules governing the declaration of these generics insure that a mapping can be established between the timing generics of a model and the corresponding SDF timing information or negative constraint delays.

```
VITAL_timing_generic_declaration ::=
    [ constant ] identifier_list ::= [ in ] type_mark [ index_constraint ] [ := static_expression ] ;
```

A timing generic is characterized by its name and its type. The naming conventions (see 4.3.2.1.1) communicate the kind of timing information specified, as well as the port(s) or delay path(s) to which the timing information applies. The type of a timing generic (see 4.3.2.1.2) indicates which of a variety of forms the associated timing value takes.

A VITAL-compliant description may declare any number of timing generics. There are no required timing generics.

*Examples:*

```
tperiod_Clk : VITALDelayType := 5 ns;
```

```

1      tpd_Clk_Q  : VITALDelayType01 := (tr01 => 2 ns, tr10 => 3 ns);
      tpd_D      : VITALDelayType01Z := ( 1 ns, 2 ns, 3 ns, 4 ns, 5 ns, 6 ns )

```

NOTE—The value of a backannotation timing generic is set during the backannotation phase; however, if negative timing constraints are in effect, its value may be adjusted during the subsequent negative constraint calculation phase.

#### 4.3.2.1.1 Timing generic names

The name of a timing generic shall adhere to the naming conventions for timing generics. If the name of a generic does not adhere to these conventions, then the generic is not a timing generic.

The form of a timing generic name and its lexical constituents are described by lexical replacement rules similar to the replacement rules for syntactic constructs. White space is included in these rules to enhance readability; however, white space is not permitted within an identifier. Different elements used to construct names are distinguished by enclosing angle brackets, which are not themselves part of the name. If a lexical element enclosed by angle brackets does not have a replacement rule, then it corresponds to a VHDL identifier described by the text inside the angle brackets. Boldface indicates literal text. Underscores serve as connectors between constituent elements; they are also literal text.

```

20      <VITALTimingGenericName> ::=
          <VITALBackannotationGenericName>
          | <VITALNegativeConstraintGenericName>

```

```

      <VITALBackannotationGenericName> ::=
          <VITALPropagationDelayName>
25      | <VITALInputSetupTimeName>
          | <VITALInputHoldTimeName>
          | <VITALInputRecoveryTimeName>
          | <VITALInputRemovalTimeName>
          | <VITALInputPeriodName>
30      | <VITALPulseWidthName>
          | <VITALInputSkewTimeName>
          | <VITALNoChangeHoldTimeName>
          | <VITALNoChangeSetupTimeName>
35      | <VITALInterconnectPathDelayName>
          | <VITALDeviceDelayName>

```

```

      <VITALNegativeConstraintGenericName> ::=
          <VITALInternalClockDelayName>
40      | <VITALInternalSignalDelayName>
          | <VITALBiasedPropagationDelayName>

```

The name of a timing generic is constructed from a timing generic prefix and a number of other elements representing device labels, ports or signals, edges, and conditions. These various elements are combined in a fixed manner, creating three distinct sections of the name: the timing generic prefix, the timing generic port specification, and the timing generic suffix.

A *timing generic prefix* is a lexical element that serves as the beginning of the VHDL simple name of a timing generic. It identifies the kind of timing information that the generic represents, which in turn determines whether the generic is a backannotation timing generic or a negative constraint timing generic. The timing generic prefix consists of the sequence of characters preceding the first underscore in the generic name. It is an error for a model to use a timing generic prefix to begin the simple name of an entity generic that is not a timing generic.

This standard defines the following set of timing generic prefixes:

|         |         |       |           |          |
|---------|---------|-------|-----------|----------|
| tpd     | tsetup  | thold | trecovery | tremoval |
| tperiod | tpw     | tskew | tncsetup  | tnchold  |
| tipd    | tdevice | tcd   | tisd      | tbpd     |

The *timing generic port specification* identifies the port(s) with which the timing data is associated. It may contain both port and instance names. A port that is referenced in a timing generic port specification is said to be *associated* with that timing generic.

The discussion of timing generic names associates timing generics with entity ports; however, a model may use a signal or some other item in place of an entity port. If the port name extracted from a timing generic port specification does not denote a port on the entity, then no assumptions are made about the item denoted by the port name, and no consistency checks are performed between the timing generic and the named item.

Backannotation and negative constraint calculation require the determination of the name(s) of the port(s) associated with a particular timing generic. A port name is *extracted* from the port specification portion of a timing generic name by taking the lexical element corresponding to that port (a sequence of characters that constitute a VHDL identifier, delimited by underscores), as defined by the naming conventions for that sort of a timing generic.

The name of a timing generic may contain a *timing generic suffix* that corresponds to a combination of SDF constructs representing conditions and edges. The forms of these SDF-related suffixes are described by the following rules:

```
<SDFSimpleConditionAndOrEdge> ::=
    <ConditionName>
    | <Edge>
    | <ConditionName>_<Edge>

<SDFFullConditionAndOrEdge> ::=
    <ConditionNameEdge> [ _<SDFSimpleConditionAndOrEdge> ]

<ConditionName> ::=
    simple_name

<Edge> ::=
    posedge
    | negedge
    | 01
    | 10
    | 0z
    | z1
    | 1z
    | z0

<ConditionNameEdge> ::=
    [ <ConditionName>_ ] <Edge>
    | [ <ConditionName>_ ] noedge
```

A condition name is a lexical element that identifies a condition associated with the timing information. The condition name may be mapped to a corresponding condition expression in an SDF file according to the mapping rules described in 5.2.7.3.2.

An edge identifies an edge associated with the timing information. The edge may be mapped to an edge name specified in an SDF file using the mapping rules described in 5.2.7.3.1.

NOTE—It is assumed that the names in timing generic port specifications will generally denote entity ports; however, a model may instead name other items that may or may not be visible from the enclosing entity declaration (internal signals, for instance). If a port name in a timing generic port specification does not denote a port on the entity, then there are no requirements for consistency between the timing generic and the named item (in fact, the named item does not even have to exist); hence, no consistency checks are performed. A tool that processes VITAL-compliant models may choose to issue a warning in this case.

#### 4.3.2.1.2 Timing generic subtypes

The type mark in the declaration of a timing generic shall denote a VITAL delay type or subtype. These are discussed in 9.1.

If each port name in the port specification of a timing generic name denotes an entity port, then the type and constraint of the timing generic shall be consistent with those of the associated port(s). This consistency is defined as follows:

- If the timing generic is associated with a single port and that port is scalar, then the type of the timing generic shall be a scalar form of delay type. If the timing generic is associated with two scalar ports, then the type of the timing generic shall be a scalar form of delay type.
- If a timing generic is declared to be of a vector form of delay type, then it represents delays associated with one or more vector ports. If such a timing generic is associated with a single port and that port is vector, then the type of the timing generic shall be a vector form of delay type, and the constraint on the generic shall match that on the associated port. If the timing generic is associated with two ports, one or more of which is vector, then the type of the timing generic shall be a vector form of delay type, and the length of the index range of the generic shall be equal to the product of the number of scalar subelements in the first port and the number of scalar subelements in the second port.

NOTE—These consistency requirements between timing generic and port(s) do not apply if the port specification in the timing generic identifies an item that is not an entity port. In this case the model assumes responsibility for the appropriate type and constraint for the timing generic.

#### 4.3.2.1.3 Timing generic specifications

Each form of timing generic represents a particular kind of timing information. Additional restrictions on the name and type or subtype may be imposed on generics representing a particular kind of timing information. A description of the acceptable forms for a particular kind of timing generic is provided in the subclause describing that kind of timing generic.

In the following discussion, an *input port* is a VHDL port of mode IN or INOUT. An *output port* is a VHDL port of mode OUT, INOUT, or BUFFER.

##### 4.3.2.1.3.1 Propagation delay

A timing generic beginning with the prefix **tpd** is a backannotation timing generic representing propagation delay associated with the specified input-to-output delay path. Its name is of the form

```
<VITALPropagationDelayName> ::=
    tpd_<InputPort>_<OutputPort> [ _<SDFSsimpleConditionAndOrEdge> ]
```

The type of a propagation delay generic shall be a VITAL delay type (see 9.1).

#### 4.3.2.1.3.2 Input setup time

A timing generic beginning with the prefix **tsetup** is a backannotation timing generic representing the setup time between an input reference port and an input test port. Its name is of the form

```
<VITALInputSetupTimeName> ::=  
    tsetup_<TestPort>_<ReferencePort> [ _<SDDFFullConditionAndOrEdge> ]
```

The type of an input setup time generic shall be a simple VITAL delay type (see 9.1).

#### 4.3.2.1.3.3 Input hold time

A timing generic beginning with the prefix **thold** is a backannotation timing generic representing the hold time between an input reference signal and an input test signal. Its name is of the form

```
<VITALInputHoldTimeName> ::=  
    thold_<TestPort>_<ReferencePort> [ _<SDDFFullConditionAndOrEdge> ]
```

The type of an input hold time generic shall be a simple VITAL delay type (see 9.1).

#### 4.3.2.1.3.4 Input recovery time

A timing generic beginning with the prefix **trecovery** is a backannotation timing generic representing the input recovery time between an input test signal and an input reference signal (similar to a setup constraint). Its name is of the form

```
<VITALInputRecoveryTimeName> ::=  
    trecovery_<TestPort>_<ReferencePort> [ _<SDDFFullConditionAndOrEdge> ]
```

The type of an input recovery time generic shall be a simple VITAL delay type (see 9.1).

#### 4.3.2.1.3.5 Input removal time

A timing generic beginning with the prefix **tremoval** is a backannotation timing generic representing input removal time between an input test signal and an input reference signal (similar to a hold constraint). Its name is of the form

```
<VITALInputRemovalTimeName> ::=  
    tremoval_<TestPort>_<ReferencePort> [ _<SDDFFullConditionAndOrEdge> ]
```

The type of an input removal time generic shall be a simple VITAL delay type (see 9.1).

#### 4.3.2.1.3.6 Period

A timing generic beginning with the prefix **tperiod** is a backannotation timing generic representing the minimum period time. Its name is of the form

```
<VITALInputPeriodName> ::=  
    tperiod_<InputPort> [ _<SDFSsimpleConditionAndOrEdge> ]
```

If present, the edge specifier indicates the edge from which the period is measured.

The type of a period generic shall be a simple VITAL delay type (see 9.1).

**4.3.2.1.3.7 Pulse width**

A timing generic beginning with the prefix **tpw** is a backannotation timing generic representing the minimum pulse width. Its name is of the form

```
<VITALPulseWidthName> ::=
    tpw_<InputPort> [ _<SDFSsimpleConditionAndOrEdge> ]
```

The edge specifier, if present, indicates the edge from which the pulse width is measured. A **posedge** specification indicates a high pulse, and a **negedge** specification indicates a low pulse.

The type of a pulse width generic shall be a simple VITAL delay type (see 9.1).

**4.3.2.1.3.8 Input skew time**

A timing generic beginning with the prefix **tskew** is a backannotation timing generic representing skew time between a pair of signals. Its name is of the form

```
<VITALInputSkewTimeName> ::=
    tskew_<FirstPort>_<SecondPort> [ _<SDFFullConditionAndOrEdge> ]
```

The type of an input skew generic shall be a simple VITAL delay type (see 9.1).

**4.3.2.1.3.9 No-change setup time**

A timing generic beginning with the prefix **tncsetup** is a backannotation timing generic representing the setup time associated with a no-change timing constraint. Its name is of the form

```
<VITALNoChangeSetupTimeName> ::=
    tncsetup_<TestPort>_<ReferencePort> [ _<SDFFullConditionAndOrEdge> ]
```

A no-change setup time generic shall appear in conjunction with a corresponding no-change hold time generic.

The type of a no-change setup time generic shall be a simple VITAL delay type (see 9.1).

**4.3.2.1.3.10 No-change hold time**

A timing generic beginning with the prefix **tnchold** is a backannotation timing generic representing a hold time associated with a no-change time constraint. Its name is of the form

```
<VITALNoChangeHoldTimeName> ::=
    tnchold_<TestPort>_<ReferencePort> [ _<SDFFullConditionAndOrEdge> ]
```

A no-change hold time generic shall appear in conjunction with a corresponding no-change setup time generic.

The type of a no-change hold time generic shall be a simple VITAL delay type (see 9.1).

**4.3.2.1.3.11 Interconnect path delay**

A timing generic beginning with the prefix **tipd** is a backannotation timing generic representing the external wire delay to a port, or an interconnect delay that is abstracted as a simple wire delay on the port. Its name is of the form

```
<VITALInterconnectPathDelayName> ::=
    tipd_<InputPort>
```