IEEE STANDARDS ASSOCIATION

IEEE Standard for the Functional Verification Language *e*

IEEE Computer Society

Developed by the
Design Automation Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997
USA

IEEE Std 1647™-2019
(Revision of
IEEE Std 1647-2016)

IEEE Std 1647™-2019
(Revision of
IEEE Std 1647-2016)

IEEE Standard for the Functional Verification Language *e*

Developed by the

Design Automation Standards Committee
of the
IEEE Computer Society

Approved 13 June 2019

IEEE SA Standards Board

Grateful acknowledgment is made to Cadence, Inc., for permission to use the following source material:

Namespaces for Types in ***e***, Version 1.1

e Language Reference, Version 18.03, Chapter 3.11 (Defining Interface Types)

e Language Reference, Version 18.03, Chapter 2.13 (The Set Type)

e Language Reference, Version 18.03, Chapter 26 (Set Pseudo Methods)

e Language Reference, Version 18.03, Chapter 5 (Template Type)

e Language Reference, Version 18.03, Chapter 2.1.11 (Defining Custom Numeric Types)

e Language Reference, Version 18.03, Chapter 32 (Tables)

e Language Reference, Version 18.03, Chapter 34 (Annotations)

e Language Reference, Version 18.03, Chapter 28.1 (Interface For Custom Numeric Types)

Abstract: The ***e*** *functional verification language* is an application-specific programming language, aimed at automating the task of verifying a hardware or software design with respect to its specification. Verification environments written in ***e*** provide a model of the environment in which the design is expected to function, including the kinds of erroneous conditions the design needs to withstand. A typical verification environment is capable of generating user-controlled test inputs with statistically interesting characteristics. Such an environment can check the validity of the design responses. Functional coverage metrics are used to control the verification effort and gauge the quality of the design. ***e*** verification environments can be used throughout the design cycle, from a high-level architectural model to a fully realized system. A definition of the ***e*** language syntax and semantics and how tool developers and verification engineers should use them are contained in this standard.

Keywords: assertion, concurrent programming, constraint, dynamic verification, functional coverage, functional verification, IEEE 1647™, simulation, temporal logic, test generation

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

PDF: ISBN 978-1-5044-5977-8 STD23758
Print: ISBN 978-1-5044-5978-5 STDPD23758

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page, appear in all standards and may be found under the heading "Important Notices and Disclaimers Concerning IEEE Standards Documents." They can also be obtained on request from IEEE or viewed at http://standards.ieee.org/ipr/disclaimers.html.

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents (standards, recommended practices, and guides), both full-use and trial-use, are developed within IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association ("IEEE SA") Standards Board. IEEE ("the Institute") develops its standards through a consensus development process, approved by the American National Standards Institute ("ANSI"), which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed through scientific, academic, and industry-based technical working groups. Volunteers in IEEE working groups are not necessarily members of the Institute and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE Standards do not guarantee or ensure safety, security, health, or environmental protection, or ensure against interference with or from other devices or networks. Implementers and users of IEEE Standards documents are responsible for determining and complying with all appropriate safety, security, environmental, health, and interference protection practices and all applicable laws and regulations.

IEEE does not warrant or represent the accuracy or content of the material contained in its standards, and expressly disclaims all warranties (express, implied and statutory) not included in this or any other document relating to the standard, including, but not limited to, the warranties of: merchantability; fitness for a particular purpose; non-infringement; and quality, accuracy, effectiveness, currency, or completeness of material. In addition, IEEE disclaims any and all conditions relating to: results; and workmanlike effort. IEEE standards documents are supplied "AS IS" and "WITH ALL FAULTS."

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.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE should be considered the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, or be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position of IEEE.

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE. However, IEEE does not provide consulting information or advice pertaining to IEEE Standards documents. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to comments or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in revisions to an IEEE standard is welcome to join the relevant IEEE working group.

Comments on standards should be submitted to the following address:

Secretary, IEEE SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854 USA

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under U.S. and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, IEEE does not waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate fee, IEEE will grant users a limited, non-exclusive license to photocopy portions of any individual standard for company or organizational internal use or individual, non-commercial use only. To arrange for payment of licensing fees, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

Updating of IEEE Standards documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect.

Every IEEE standard is subjected to review at least every ten years. When a document is more than ten years old and has not undergone a revision process, 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.

In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit IEEE Xplore at http://ieeexplore.ieee.org or contact IEEE at the address listed previously. For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website at http://standards.ieee.org.

Errata

Errata, if any, for all IEEE standards can be accessed on the IEEE SA Website at the following URL: http://standards.ieee.org/findstds/errata/index.html. Users are encouraged to check this URL for errata periodically.

Patents

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 by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance with an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA Website at http://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses.

Essential Patent Claims may exist for which a Letter of Assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this standard was submitted to the IEEE SA Standards Board for approval, the IEEE ***e*** Functional Verification Language Working Group had the following members and observers:

Darren Galpin, *Chair and Technical Editor*
Yuri Tsoglin, *Vice-Chair*

Cristian Amitroaie
Mike Bartley
Stefan Birman
Silas McDermott
Genadi Osowiecki
Andrew Piziali
Marcus Harnisch
Yaron Kashai
Efrat Shneydor
Uwe Simm
Akash Singh

The following members of the individual balloting committee voted on this standard. Ballots may have voted for approval, disapproval, or abstention.

Robert Aiello
Heiko Ehrenberg
Darren Galpin
Randall Groves
Werner Hoelzl
Osamu Karatsu
Piotr Karocki
Yaron Kashai
Rakesh Kumar
Bansi Patel
Andrew Piziali
Stephen Schwalm
Walter Snapper
John Wegis

When the IEEE SA Standards Board approved this amendment on 13 June 2019, it had the following membership:

Gary Hoffman, *Chair*
Ted Burse, *Vice Chair*
Jean-Philippe Faure, *Past Chair*
Konstantinos Karachalios, *Secretary*

Masayuki Ariyoshi
Stephen D. Dukes
J. Travis Griffith
Guido Hiertz
Christel Hunter
Joseph L. Koepfinger*
Thomas Koshy
John D. Kulick
David J. Law
Joseph Levy
Howard Li
Xiaofeng Liu
Kevin Lu
Daleep Mohla
Andrew Myles
Annette D. Reilly
Dorothy Stanley
Sha Wei
Phil Wennblom
Philip Winston
Howard Wolfman
Feng Wu
Jingyi Zhou

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1647-2019, IEEE Standard for the Functional Verification Language ***e***.

The ***e*** *functional verification language* is an application-specific programming language aimed at the problem of verifying functional correctness of hardware and software designs. Simply stated, functional verification attempts to provide a quantitative answer to the question: How well does the design match the functional specification?

Functional correctness of chip designs grew in criticality from the mid-1980s. As design complexity kept growing, ad hoc testing methods ran out of steam and a more systematic verification approach was necessary. Manually constructed test suites, the early method of choice, became both uneconomical and ineffective when scaled up. As a result, many companies supplemented manual test suites with *pseudo-random generation* of input stimulus. Such test generation programs were typically built for a particular project or a particular architecture. They turned out to be expensive to develop and maintain, but once functional, they would clean up the design in a very thorough way.

A key observation made by Yoav Hollander, the creator of ***e***, was that verification environments of different projects have a lot in common and yet each verification environment is structured to match a particular design specification. Hollander's solution was to create a language that had verification-specific constructs as primitives and the full capabilities of a high-level language for customization. In particular, pseudo-random test generation became a built-in capability of the language. Early prototypes of the language were experimented with as early as 1993, showing significant productivity gains.

The ***e*** language was productized by Verisity, Ltd., in 1996, as part of a functional verification tool suite. The proliferation of the ***e*** language and the growing investment in ***e***-based intellectual property (IP) compelled the creation of the ***e*** *steering committee* in June of 2002, composed of individuals from Texas Instruments, Rambus, ST Microelectronics, Cisco, Intel, Axis System, ARC, and Verisity. The ***e*** *steering committee* recommended the ***e*** language be standardized through The Institute of Electrical and Electronics Engineers (IEEE). Accepting the recommendation, Verisity released the rights to the language to the IEEE in June of 2003.

The ***e*** language, in its current form, brings together concepts from many domains.

- ***e*** has a basic object-oriented (OO) programming model, with implicit memory management and single inheritance. In this ***e*** is similar to Java™.[a]
- Beyond objects, ***e*** supports *aspects*, which can be viewed as layers cutting across multiple objects. Adding an aspect to an existing program refines the program by introducing a coherent change to a plurality of objects.
- ***e*** supports constraints as object features. Constraints are used to refine object modeling. The execution model of the language involves resolving constraint systems and picking random values that would satisfy constraint systems.
- ***e*** is a strongly typed language, like Pascal and Modula.
- ***e*** has concurrency constructs and modeling blocks for hierarchical composition, similar to hardware description languages like Verilog® [b](see IEEE Std 1364™)[c] and VHDL (see IEC/IEEE 61691-1-1). Concurrency in ***e*** is synchronous, like in Esterel.
- ***e*** contains a temporal language that borrows from Linear Temporal Logic and Interval Temporal Logic.

[a] Java is a trademark of Sun Microsystems, Inc. in the United States and other countries.

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

[c] Information on references can be found in Clause 2.

— ***e*** has many built-in constructs aimed at simplifying common programming tasks; built-in support for lists and hashes; and pattern matching and string- and file-manipulation features, which are borrowed from Perl®.[d]

— The ***e*** syntax is extendable with a powerful macro capability.

This combination of concepts caters directly to the needs of verification engineers, removing the need to cobble together multiple components in different languages.

As with any programming language, the source of ingenuity is with the programmer. Verification engineers need sound methodologies, creativity, an inquisitive mind, and a keen eye for poorly specified aspects. Yet experience with ***e*** shows that when put to good use, the ***e*** language fosters productivity and quality resul

[d] Perl is a registered trademark of Perl, Inc.

Contents

1. Overview ... 16

1.1 Introduction ... 16
1.2 Scope ... 16
1.3 Purpose ... 16
1.4 Verification environments ... 16
1.5 Basic concepts relating to this standard ... 17
1.6 Conventions used ... 23
1.7 Use of color in this standard ... 25
1.8 Contents of this standard ... 25

2. Normative references ... 27

3. Definitions, acronyms, and abbreviations ... 28

3.1 Definitions ... 28
3.2 Acronyms and abbreviations ... 29

4. ***e*** basics ... 31

4.1 Overview ... 31
4.2 Lexical conventions ... 31
4.3 Syntactic elements ... 38
4.4 Struct hierarchy and name resolution ... 44
4.5 Ranges ... 50
4.6 Operator precedence ... 51
4.7 Evaluation order of expressions ... 52
4.8 Bitwise operators ... 53
4.9 Boolean operators ... 54
4.10 Arithmetic operators ... 56
4.11 Comparison operators ... 57
4.12 String matching ... 63
4.13 Extraction and concatenation operators ... 66
4.14 Scalar modifiers ... 69
4.15 Parentheses ... 70
4.16 list.method() ... 70
4.17 Special-purpose operators ... 71

5. Data types ... 76

5.1 Overview ... 76
5.2 Scalar data types ... 76
5.3 Untyped expressions ... 84
5.4 Assignment rules ... 85
5.5 Real data type ... 88
5.6 Precision rules for numeric operations ... 90
5.7 Automatic type casting ... 92
5.8 Defining and extending scalar types ... 93
5.9 Referring to types in generic code ... 96
5.10 Type-related constructs ... 97

6. Structs, subtypes, and fields ... 105

6.1 Overview ... 105
6.2 Structs overview ... 105
6.3 Defining structs: struct ... 106
6.4 Extending structs: extend type ... 107
6.5 Restrictions on inheritance ... 108
6.6 Extending subtypes ... 108
6.7 Creating subtypes with when ... 108
6.8 Extending when subtypes ... 110
6.9 Defining fields: field ... 111
6.10 Defining list fields ... 114
6.11 Projecting list of fields ... 116
6.12 Defining attribute fields ... 116
6.13 Defining interface types: interface ... 117

7. Units ... 119

7.1 Overview ... 119
7.2 Defining units and fields of type unit ... 122
7.3 Unit attributes ... 125
7.4 Predefined methods of any_unit ... 127
7.5 Unit-related predefined methods of any_struct ... 130
7.6 Unit-related predefined routines ... 132

8. Template types ... 134

8.1 Overview ... 134
8.2 Defining a struct/unit template type ... 135
8.3 Defining a template interface type ... 138
8.4 Defining a template numeric type ... 139
8.5 Template parameters ... 140
8.6 Extending a template struct ... 142
8.7 Instantiating a template type ... 144
8.8 Template subtype instances ... 145

9. ***e*** ports ... 146

9.1 Overview ... 146
9.2 Introduction to ***e*** ports ... 146
9.3 Using simple ports ... 147
9.4 Using buffer ports ... 149
9.5 Using event ports ... 150
9.6 Using method ports ... 151
9.7 Defining and referencing ports ... 153
9.8 Port attributes ... 161
9.9 Buffer port methods ... 173
9.10 MVL methods for simple ports ... 175
9.11 Global MVL routines ... 186
9.12 Comparative analysis of ports and tick access ... 189
9.13 ***e*** port binding declaration and methods ... 190
9.14 Transaction-level modeling interface ports in ***e*** ... 197
9.15 TLM sockets in ***e*** ... 205

10. Constraints and generation ... 212

10.1 Overview ... 212
10.2 Types of constraints ... 212
10.3 Generation concepts ... 213
10.4 Type constraints ... 230
10.5 Defining constraints ... 232
10.6 Invoking generation ... 239

11. Temporal struct members ... 241

11.1 Events ... 241
11.2 on ... 245
11.3 on event-port ... 246
11.4 expect | assume ... 247
11.5 Procedural API for temporal operators on event and expect struct members ... 249

12. Temporal expressions ... 263

12.1 Overview ... 263
12.2 Temporal operators and constructs ... 265
12.3 Success and failure of a temporal expression ... 280

13. Time-consuming actions ... 282

13.1 Overview ... 282
13.2 Synchronization actions ... 282
13.3 Concurrency actions ... 284
13.4 State machines ... 285

14. Coverage constructs ... 290

14.1 Overview ... 290
14.2 Defining coverage groups: cover ... 290
14.3 Defining basic coverage items: item ... 292
14.4 Defining cross coverage items: cross ... 297
14.5 Defining transition coverage items: transition ... 299
14.6 Extending coverage groups: cover ... using also ... is also ... 301
14.7 Extending coverage items: item ... using also ... 302
14.8 Coverage API ... 303
14.9 Coverage methods for the covers struct ... 309

15. Macros ... 317

15.1 Overview ... 317
15.2 define-as statement ... 318
15.3 define-as-computed statement ... 318
15.4 Match expression structure ... 319
15.5 Interpretation of match expressions ... 321
15.6 Macro expansion code ... 322

16. Print, checks, and error handling ... 325

16.1 Overview ... 325

16.2 print 325
16.3 Handling DUT errors 325
16.4 Handling user errors 331
16.5 Handling programming errors: assert 333

17. Methods 334

17.1 Overview 334
17.2 Rules for defining and extending methods 334
17.3 Invoking methods 343
17.4 Parameter passing 347
17.5 Using the C interface 349

18. Creating and modifying ***e*** variables 351

18.1 About ***e*** variables 351
18.2 var 351
18.3 = ... 352
18.4 op= 352
18.5 <= 353
18.6 rgf= 355

19. Packing and unpacking 356

19.1 Basic packing 356
19.2 Predefined pack options 359
19.3 Customizing pack options 360
19.4 Packing and unpacking specific types 360
19.5 Implicit packing and unpacking 366

20. Control flow actions 367

20.1 Conditional actions 367
20.2 Iterative actions 369
20.3 File iteration actions 374
20.4 Actions for controlling the program flow 375

21. Importing and preprocessor directives 377

21.1 Overview 377
21.2 Importing ***e*** modules 377
21.3 #ifdef, #ifndef 378
21.4 #define 379
21.5 #undef 380

22. Encapsulation constructs 381

22.1 Overview 381
22.2 package: package-name 381
22.3 package: type-declaration 381
22.4 package | protected | private: struct-member 382
22.5 Scope operator (::) 383

23. Simulation-related constructs 384

23.1 Overview ... 384
23.2 force ... 384
23.3 release ... 384
23.4 Tick access: 'hdl-pathname' ... 385

24. Messages ... 386

24.1 Overview ... 386
24.2 Message model ... 386
24.3 Message execution ... 386
24.4 Structured debug messages ... 387
24.5 message and messagef ... 392
24.6 Tag ... 393
24.7 Verbosity ... 393
24.8 Predefined type sdm_handler ... 394
24.9 Messages interface ... 396

25. Sequences ... 418

25.1 Overview ... 418
25.2 Sequence statement ... 420
25.3 do sequence action ... 422
25.4 Sequence struct types and members ... 423
25.5 BFM-driver-sequence flow diagrams ... 428

26. Tables ... 433

26.1 The table construct ... 433
26.2 table ... 433
26.3 Table operators ... 435

27. List pseudo-methods library ... 438

27.1 Pseudo-methods overview ... 438
27.2 Using list pseudo-methods ... 438
27.3 Pseudo-methods to modify lists ... 438
27.4 General list pseudo-methods ... 447
27.5 Math and logic pseudo-methods ... 463
27.6 List CRC pseudo-methods ... 466
27.7 Keyed list pseudo-methods ... 467

28. Predefined methods library ... 470

28.1 Ovefview ... 470
28.2 Predefined methods of sys ... 470
28.3 Predefined methods of any_struct ... 470
28.4 Methods and predefined attributes of unit any_unit ... 475
28.5 Set pseudo-methods ... 475
28.6 Other pseudo-methods ... 482
28.7 Coverage methods ... 483

29. Predefined routines library ... 485

29.1 Overview ... 485

29.2 Deep copy and compare routines ... 485
29.3 Integer arithmetic routines ... 489
29.4 Real arithmetic routines ... 493
29.5 bitwise_op() ... 494
29.6 get_all_units() ... 495
29.7 String routines ... 495
29.8 Output routines ... 503
29.9 Operating system interface routines ... 505
29.10set_config() ... 508
29.11Randomization routines ... 509
29.12Simulation-related routines ... 509
29.13Range-generated field routines ... 511

30. Predefined file routines library ... 513

30.1 Overview ... 513
30.2 File names and search paths ... 513
30.3 File handles ... 513
30.4 Low-level file methods ... 513
30.5 General file routines ... 518
30.6 Reading and writing structs ... 524

31. Predefined interfaces library ... 527

31.1 Interfaces for custom numeric types ... 527
31.2 base_numeric_i ... 527
31.3 numeric_i ... 529
31.4 Implementing numeric_i ... 533
31.5 Predefined types related to custom numeric interface ... 534

32. Predefined APIs and utilities library ... 535

32.1 Reflection API ... 535
32.2 instance_iterator object hierarchy traversal API ... 569
32.3 Procedural API for tables ... 572

33. Predefined resource sharing control structs ... 578

33.1 Overview ... 578
33.2 Semaphore methods ... 578
33.3 How to use the semaphore struct ... 579

34. Intellectual property protection ... 583

34.1 Overview ... 583
34.2 Encryption ... 583
34.3 Decryption ... 584
34.4 Reflection API ... 584
34.5 Encryption targets ... 584

35. Annotations ... 585

35.1 Overview ... 585
35.2 annotation @annotation-type-name ... 585

35.3 Using annotations 586
35.4 Annotation API 588

Annex A (informative) Bibliography 591

Annex B (normative) Source code serialization 592

Annex C (informative) Comparison of when and like inheritance 600

Annex D (normative) Name spaces 608

Annex E (informative) Reflection API examples 616

Annex F (informative) Encryption targets 620

IEEE Standard for the Functional Verification Language *e*

1. Overview

1.1 Introduction

This clause explains the scope and purpose of this standard; gives an overview of the basic concepts, major semantic components, and conventions used in this standard; and summarizes its contents.

1.2 Scope

This standard defines the ***e*** functional verification language. This standard aims to serve as an authoritative source for the definition of (a) syntax and semantics of ***e*** language constructs (b) the ***e*** language interaction with standard simulation languages (c) ***e*** language libraries.

1.3 Purpose

This standard serves the community involved with functional verification of electronic designs using the ***e*** language. It provides an implementation independent definition of the ***e*** language and facilitates the development of ***e*** language based design automation tools.

1.4 Verification environments

Electronic systems are integrated circuits (ICs), boards, or modules combining multiple ICs together, along with optional embedded processors and software components. Electronic systems are built to *specifications* that anticipate the environment in which such systems are expected to function and define the expected system functionality. *Functional verification* measures how well a system meets its specification. Even with moderately complex systems this question cannot be answered by inspection. For all modern electronic systems, a sophisticated *verification process* needs to accompany the design process to ensure compliance with the specification.

Many electronic design automation (EDA) tools are used to carry out the functional verification process. The most prominent functional verification method, used to verify virtually all system designs, is called *dynamic verification* or *simulation-based verification*. Simulation-based verification makes use of a functional model of the system being designed. The functional model is *simulated* in the context of a mock-up of the anticipated system environment. This mock-up is called the *verification environment*.