IEC 62526

Edition 1.0 2007-11

INTERNATIONAL STANDARD

IEEE 1450.1™

Standard for Extensions to Standard Test Interface Language (STIL) for Semiconductor Design Environments

IEC 62526:2007(E) IEEE Std. 1450.1-2005

AVA Documents
Document Sharing Hub

Full document details: IEC 62526 2007

THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2007 IEEE

All rights reserved. IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by the Institute of Electrical and Electronics Engineers, Inc.

Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the IEC Central Office.

Any questions about IEEE copyright should be addressed to the IEEE. Enquiries about obtaining additional rights to this publication and other information requests should be addressed to the IEC or your local IEC member National Committee.

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

The Institute of Electrical and Electronics Engineers, Inc
3 Park Avenue
US-New York, NY10016-5997
USA
Email: stds-info@ieee.org
Web: www.ieee.org

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

IEC 62526

Edition 1.0 2007-11

INTERNATIONAL STANDARD

IEEE 1450.1 TM

Standard for Extensions to Standard Test Interface Language (STIL) for Semiconductor Design Environments

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

PRICE CODE XF

ICS 25.040

ISBN 2-8318-9348-8

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

CONTENTS

FOREWORD....5

IEEE Introduction....8

1. Overview....9

1.1 Scope....10
1.2 Purpose....11

2. Definitions, acronyms, and abbreviations....11

2.1 Definitions....11
2.2 Acronyms and abbreviations....12

3. Structure of this standard....12

4. STIL syntax description....13

4.1 Reserved words....13
4.2 Reserved characters....14
4.3 Reserved UserFunctions....15
4.4 Signal and group name characteristics....15
4.5 STIL name spaces and name resolution....16

5. Expressions....17

5.1 Constant and variable expressions....17
5.2 Expression delimiters—single quotes and parentheses....17
5.3 Arithmetic expressions—integer, real, time, boolean....19
5.4 Pattern data expressions....20
5.5 Expression processing....21
5.6 Boolean—boolean_expr....26
5.7 *Integers—integer_expr*....26
5.8 Logic expressions—logic_expr....27
5.9 Real expressions—real_expr....28
5.10 Addition to timing expressions—time_expr....29
5.11 SignalVariables—sigvar_expr....30
5.12 Formal parameters in procedures and macros....32
5.13 Integer lists—integer_list....32

6. Statement structure and organization of STIL information....33

7. STIL statement....33

7.1 STIL syntax....34
7.2 STIL example....34

8. UserKeywords statement....34

8.1 UserKeywords syntax....34
8.2 UserKeywords example....34

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

9. Variables block ... 35

9.1 Variables block syntax ... 35
9.2 Variables example ... 37
9.3 Variables scoping ... 37
9.4 Variables synchronizing ... 39

10. Signals block ... 40

10.1 Signals block syntax ... 40
10.2 Signals example ... 40
10.3 Bracketed signal notation enhancement ... 40

11. SignalGroups block ... 43

11.1 SignalGroups syntax ... 43
11.2 SignalGroups, WFCMap, and Variables example ... 43
11.3 Default WFCMap attribute value ... 44
11.4 Defining indexed signal groups ... 44

12. PatternBurst block ... 45

12.1 PatternBurst syntax ... 45
12.2 PatternBurst example ... 47
12.3 Tiling and synchronization of patterns ... 48
12.4 If and While statements ... 50

13. Timing block and WaveformTable block ... 51

13.1 Additional domain specification ... 51
13.2 CompareSubstitute operation—s, S ... 51

14. ScanStructures block ... 52

14.1 ScanStructures syntax ... 52
14.2 Scan cell naming—cell_ref, chain_ref, cell_group, chain_group ... 55
14.3 Scoping rules for ScanStructure blocks ... 56
14.4 Example indexed list of scan cells ... 57
14.5 Example of ScanChainGroups and ActiveScanChain ... 57
14.6 Scan chain segments and cell groups ... 59

15. Pattern data ... 60

15.1 Data content read back—\C, \D, \E, \S, \U, \W ... 61
15.2 Vector data mapping and joining—\m, \j ... 63
15.3 Specifying event data in a pattern—\e ... 65
15.4 Using expressions within pattern data ... 66

16. Pattern statements ... 67

16.1 Additional Pattern syntax ... 68
16.2 Vector data constraints—F, E ... 69
16.3 Shift and LoopData statements ... 70
16.4 Loop statement using an integer expression ... 72
16.5 MergedScan function ... 73

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

17. Procedure and macro data substitution ... 73

17.1 Nested procedure and macro cells ... 73
17.2 Passing parameters to variables ... 74
17.3 Default value of formal parameters ... 75
17.4 Data substitution using WFCConstant and SignalVariable ... 75

18. Environment block ... 77

18.1 Environment syntax ... 77
18.2 MAP_STRING syntax ... 79
18.3 NameMaps example ... 79
18.4 Compact scan-cell mapping using InheritNameMap ... 81

19. Pragma block ... 82

19.1 Pragma syntax ... 82

20. PatternFailReport ... 82

20.1 PatternFailReport syntax ... 83
20.2 PatternFailReport example ... 84

Annex A (informative) Glossary ... 84

Annex B (informative) Signal mapping using SignalVariables ... 87

Annex C (informative) Using logic expression with signals ... 91

Annex D (informative) Using boolean expressions in patterns ... 92

Annex E (informative) Variables and expressions in algorithmic patterns ... 93

Annex F (informative) Using AllowInterleave ... 95

Annex G (informative) Vector data mapping using \m ... 98

Annex H (informative) Vector data joining using \j ... 101

Annex I (informative) Block data collection ... 104

Annex J (informative) Using Fixed and Equivalent statements ... 106

Annex K (informative) Independent parallel patterns ... 108

Annex L (informative) Applications using new ScanStructures syntax ... 110

Annex M (informative) BreakPoints using MergedScan() function ... 114

Annex N (informative) Labels and X statements for diagnostic feedback ... 117

Annex O (informative) Use of STIL.1 for specific applications ... 120

Annex P (informative) Bibliography ... 122

Annex Q (informative) List of participants ... 123

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

AVA Documents
Document Sharing Hub

Full document details: IEC 62526 2007

INTERNATIONAL ELECTROTECHNICAL COMMISSION

STANDARD FOR EXTENSIONS TO STANDARD TEST INTERFACE LANGUAGE (STIL) FOR SEMICONDUCTOR DESIGN ENVIRONMENTS

FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
5) IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with an IEC Publication.
6) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC/IEEE 62526 has been processed through Technical Committee 93: Design automation.

The text of this standard is based on the following documents:

IEEE Std	FDIS	Report on voting
1450.1(2005)	93/248/FDIS	93/259/RVD

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table.

The committee has decided that the contents of this publication will remain unchanged until the maintenance result date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: IEC 62526 2007

IEC/IEEE Dual Logo International Standards

This Dual Logo International Standard is the result of an agreement between the IEC and the Institute of Electrical and Electronics Engineers, Inc. (IEEE). The original IEEE Standard was submitted to the IEC for consideration under the agreement, and the resulting IEC/IEEE Dual Logo International Standard has been published in accordance with the ISO/IEC Directives.

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEC/IEEE Dual Logo International Standard is wholly voluntary. The IEC and IEEE disclaim liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEC or IEEE Standard document.

The IEC and IEEE do not warrant or represent the accuracy or content of the material contained herein, and expressly disclaim any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEC/IEEE Dual Logo International Standards documents are supplied “AS IS”.

The existence of an IEC/IEEE Dual Logo International 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 IEC/IEEE Dual Logo International 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.

In publishing and making this document available, the IEC and IEEE are not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Neither the IEC nor IEEE is undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEC/IEEE Dual Logo International Standards or IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

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 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 societies and Standards Coordinating 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 for revision of IEC/IEEE Dual Logo International Standards are welcome from any interested party, regardless of membership affiliation with the IEC or IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA and/or General Secretary, IEC, 3, rue de Varembé, PO Box 131, 1211 Geneva 20, Switzerland.

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; +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.

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 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.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450™-1999) for Semiconductor Design Environments

Sponsor

Test Technology Standards Committee
of the
IEEE Computer Society

Approved 9 June 2005

IEEE-SA Standards Board

Abstract: Standard Test Interface Language (STIL) provides an interface between digital test generation tools and test equipment. Extensions to the test interface language (contained in this standard) are defined that (1) facilitate the use of the language in the design environment and (2) facilitate the use of the language for large designs encompassing subdesigns with reusable patterns.
Keywords: advanced scan architecture, core, environment, fail feedback, lockstep, parallel patterns, parameterized data, pattern tiling, pragma, signal variable, system on chip (SoC), test protocol

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

IEEE Introduction

The Standard Test Interface Language (STIL) was initially developed by an ad hoc consortium of automatic test equipment vendors (ATE), electronic design automation vendors (EDA), and integrated circuit (IC) manufacturers to address the lack of a common solution for transferring digital test data from the generation environment to the test equipment.

The scope of the initial STIL standard was limited to satisfy the basic needs of pattern definition. Additional capabilities are developed as separate projects resulting in separate (dot) extensions to the initial STIL standard. The scope of this extension is defined in 1.1 and is primarily to address design needs.

Whereas the initial STIL standard was developed by reviewing many languages already in existence in the industry, this standard has been developed by inventing new capabilities in support of new device designs. The new language constructs have been added such that they do not alter in any way the initial definition of STIL, yet are syntactically compatible with the initial STIL language.

Much of the work to develop and validate these extensions has been done by prototyping on the part of the contributing companies.

Notice to users

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

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 with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents or patent applications for which a license may be required to implement an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

STANDARD FOR EXTENSIONS TO STANDARD TEST INTERFACE LANGUAGE (STIL) (IEEE Std 1450™-1999) FOR SEMICONDUCTOR DESIGN ENVIRONMENTS

1. Overview

STIL is an evolving standard being developed in support of various needs for interfacing between test generation tools and test equipment. IEEE Std 1450-1999 (STIL.0) [B3][1] forms the basis for this evolution. New "dot" standards (like this one) are being developed to address specific needs beyond STIL.0.

This (STIL.1) standard addresses design-related aspects of digital test data. This standard can also be viewed as the addition of advanced features to the STIL.0 baseline to allow for the usage of STIL in more complex applications, while leaving the basic standard unchanged as a vehicle for transmitting basic test data. The following is a brief overview of the new features in STIL.1 to support advanced applications such as (1) embedded cores,[2] (2) families of test patterns, (3) mapping to automated test equipment (ATE) systems,[3] (4) mapping to simulation, and (5) devices with advanced design for test (DFT). Please see Annex O for a list of specific statements for each of these features.

Environment mapping: Data for a device exist in many forms and in many other software environments. Examples include (1) simulation environment, (2) static analysis environment, (3) specific ATE system environment. The STIL Environment block is a new mechanism to relate STIL data to these other environments. No assumptions, expectations, or limitations are imposed on the other environments. It is just a way of relating one to the other.

Parameterized data: Much of STIL data are declarative in nature (i.e., it defines various static attributes of a device or pattern set). The addition of constant declarations, IntegerConstant and WFCConstant, allows a data set to be created that applies to a family of devices.

Complex test protocol definition: Test protocol definitions are usually contained in STIL procedures or MacroDefs and are used to specify the application of a series of data values to a device. STIL.0 supports scan chain data passing and simple WaveformCharacter (WFC) data passing via the # and % characters. STIL.1 enhances this capability by allowing the use of data substitution from SignalVariables and integer-

[1]The numbers in brackets correspond to those of the bibliography in Annex P.

[2]This standard contains syntax in support of embedded cores. See IEEE Std 1450.6™-2005 (Core Test Language) [B5] for the complete specification.

[3]This standard contains syntax in support of ATE systems. See IEEE P1450.3™ (Test Resource Constraints) [B4] for the complete specification.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: IEC 62526 2007

Complex test protocol definition: Test protocol definitions are usually contained in STIL procedures or MacroDefs and are used to specify the application of a series of data values to a device. STIL.0 supports scan chain data passing and simple WaveformCharacter (WFC) data passing via the # and % characters. STIL.1 enhances this capability by allowing the use of data substitution from SignalVariables and integer-expressions. STIL.1 also enhances the functionality of Loops and Vectors and adds If/While decisions on pattern statements. These capabilities are needed for BIST, embedded cores, and various test access mechanisms.

Advanced scan architecture: Advanced DFT techniques require additional capabilities beyond what is defined in STIL.0, which includes multistate scan cells, reconfigurable scan-chains, and scan-chain indexing.

Run-time pattern decisions: The If, Else, While, and LoopData are new STIL.1 constructs that have been added for specification of pattern activity. These statements are needed in the specification of patterns to be run in the simulation environment. Although there is no standardization among ATE systems on run-time instructions for pattern execution, it is anticipated that restricted versions of these statements will be incorporated into ATE test patterns.

Pattern burst options: New variations on the PatternBurst have been added to allow for patterns running in parallel, patterns running in LockStep, and patterns that can be reordered. For parallel pattern execution, the specification for how the patterns fit together can be specified with the Fixed and Extend statements.

Enhanced user extensibility: The UserKeyword extensibility defined in STIL.0 has been extended to allow keywords to be defined on a per-block-type basis.

Signal relationships: Additional syntax is provided to allow the specification of relationships between signals. This process is preformed via \m to map WFCs to another WFC, \j to join WFCs, Extend to define behavior of signals beyond the bounds of a given pattern, and Fixed to restrict any further changes to signals within a pattern.

Fail feedback: Three new features are added to facilitate the processing of failure data from an ATE system back to design tools. The first is the X or cross-reference statement that allows the specification of where in a pattern/vector sequence a failure occurs. The second is the FailFeedback block for reporting fails. The third is the S/s timing event that allows for the specification of a data capture protocol for the purpose of capturing bulk fail data for processing.

1.1 Scope

Structures are defined in STIL to support usage as semiconductor simulation stimulus, including (1) mapping signal names to equivalent design references, (2) interface between scan and built-in self test (BIST) and the logic simulation, (3) data types to represent unresolved states in a pattern, (4) parallel or asynchronous pattern execution on different design blocks, and (5) expression-based conditional execution of pattern constructs.

Structures are defined in STIL to support the definition of test patterns for sub-blocks of a design[4] (i.e., embedded cores) such that these tests can be incorporated into a complete higher level device test.

Structures are defined in STIL to relate fail information from device testing environments back to original stimulus and design data elements.

[4]Syntax in this document that is used in the definition of patterns for sub-blocks is summarized in Annex O.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

1.2 Purpose

The STIL language definition is enhanced to support the usage of STIL in the design environment, which includes extending the execution concept to support STIL as a stimulus language, to allow STIL to be used as an intermediate form of data, and to allow STIL to capture design information needed to port simulation data to device test environments.

In addition, define extensions to support the definition of subelement tests and to define the mechanisms to integrate those tests into a complete device test. This effort is to be performed in conjunction with IEEE Std 1500™-2005 [B6] and IEEE P1450.6 [B5], which are defining standards for the definition and integration of embedded cores.

Finally, define the constructs necessary to correlate test failure information back to the design environment, to allow debug and diagnosis operations to be performed based on failure information in STIL format.

2. Definitions, acronyms, and abbreviations

2.1 Definitions

For the purposes of this standard, the following terms and definitions apply. Additional terminology specific to this standard is found in Annex A. *The Authoritative Dictionary of IEEE Standards Terms* [B1] should be referenced for terms not defined in this clause.

2.1.1 automated test equipment (ATE): It refers to a tester that is capable of interfacing to a semiconductor device and executing test pattern data that is imported from a STIL file/stream.

2.1.2 built-in self-test (BIST): A design practice in which test logic is incorporated into the circuitry of a semiconductor device. This circuitry may provide completely autonomous testing of a device (i.e., without any requirement of a tester). It may be such that stimulation by an external tester is required; however, the STIL file may be substantially different from a device without this circuitry incorporated.

2.1.3 core: A component or module that contains separately developed functionality, integrated into a chip to provide additional overall functionality. *See also:* **System on Chip.**

2.1.4 electronic design automation (EDA): The set of software tools that are used for the design and creation of semiconductor chips. It includes the software tools that create the test patterns for the chips, which are often referred to as automated test pattern generators (ATPGs).

2.1.5 Standard Test Interface Language (STIL): The set of IEEE standards, including IEEE Std 1450-1999, and all dotted extensions, including this one.

2.1.6 System on Chip (SoC): An integrated circuit containing modules that are designed/integrated such that they can be tested independently and have associated test patterns for each module.

2.1.7 WaveformCharacter (WFC): A symbol used for referencing waveforms.

NOTE—See Annex A.[5]

[5]Notes in text, tables, and figures are given for information only, and do not contain requirements needed to implement the standard.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

2.1.8 WaveformTable (WFT): An STIL block statement used to define waveforms across multiple signals and WFCs.

NOTE—See Annex A.

2.1.9 WaveformGenerationLanguage (WGL): A proprietary standard that was, in part, used as the basis for STIL.0.

2.2 Acronyms and abbreviations

ATE	automated test equipment
BIST	built-in self-test
DFT	design for test
DUT	device under test
EDA	electronic design automation
STIL	Standard Test Interface Language
STIL.0	IEEE Std 1450-1999 [B3]
STIL.1	this standard
WFC	WaveformCharacter
WFT	WaveformTable
WGL	WaveformGenerationLanguage

3. Structure of this standard

This standard is an adjunct to STIL.0. The conventions established and defined in STIL.0 are used in this standard and are included verbatim.

Many clauses in this standard add additional constructs to existing clauses in STIL.0 and are so identified in the title. The Environment block is a new construct introduced in this standard. All clauses in this standard are normative. Example code is provided within each clause. More complete examples are provided in the annexes, which are informative.

The following is a copy of the conventions as defined in STIL.0 and adhered to in this standard.

Different fonts are used as follows:

— SMALL CAP TEXT indicates user data.

— `Courier text` indicates code examples.

In the syntax definitions

a) SMALL CAP TEXT indicates user data.

b) **Bold text** indicates keywords.

c) *Italic text* references metatypes.

d) () indicates optional syntax that may be used zero or one time.

e) ()+ indicates syntax that may be used one or more times.

f) ()* indicates optional syntax that may be used zero or more times.

g) <> indicates multiple-choice arguments or syntax.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

In the syntax explanations, the verb “shall” indicates mandatory requirements. The meaning of a mandatory requirement varies for different readers of the standard:

— To developers of tools that process STIL (readers), “shall” denotes a requirement that the standard imposes. The resulting implementation is required to enforce this requirement and issue an error if the requirement is not met by the input.

— To developers of STIL (writers), “shall” denotes mandatory characteristics of the language. The resulting output must conform to these characteristics.

— To the users of STIL, “shall” denotes mandatory characteristics of the language. Users may depend on these characteristics for interpretation of the STIL source.

The language definition clauses contain statements that use the phrase “it is an error” and “it may be ambiguous.” These phrases indicate improperly defined STIL information. The interpretation of these phrases will differ for the different readers of this standard in the same way that “shall” differs, as identified here in the dashed list.

4. STIL syntax description

This clause defines extensions to STIL.0, Clause 6.

All constructs and restrictions for STIL.0, Clause 6 are in effect here, with the following additions:

— Additional STIL reserved words are defined (see Table 1) for the top-level blocks specified within this standard.

— Additional STIL reserved characters are defined (see Table 2) for the new characters specified within this standard.

— Additional definition of signal and group naming, as well as name space resolution, is provided in this clause.

— Extensions to the expression environment are defined in this standard. Expression definitions (which are part of Clause 5 in STIL.0) are now specified in a separate clause (Clause 6 in this standard).

4.1 Reserved words

Table 1 lists all STIL reserved words defined by this standard. Only top-level block names that are defined in this standard are added to the STIL reserved word list. New keywords that appear inside of top-level blocks are not restricted from usage in other contexts outside of the definition of that keyword. No change to reserved words as defined in STIL.0 is made by this standard.

Table 1—Additions to STIL reserved words

Environment
PatternFailReport, Pragma
Variables

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

4.2 Reserved characters

Several reserved characters identified in STIL.0 are applied in additional contexts for this standard. Table 2 lists additional STIL reserved characters defined in this standard as well as additional contexts of previously identified reserved characters. No change to reserved characters as defined in STIL.0 is made by this standard. New reserved characters are so indicated in column two. All other characters in this table have extended capability beyond that defined in STIL.0.

Table 2—Additions to STIL reserved characters

Char	New in STIL.1	Usage
!		**exclamation (NOT sign)**: An inversion operator in expressions
%		**percent sign**: Is used as the modulus in expressions and for parameter passing to procedures and macros
( )		**parentheses**: Are used in expressions
*		**multiply**: Is used in expressions
+		**add**: Is used in expressions
-		**subtract**: Is used in expressions
&	YES	**and**: Is used in expressions
|	YES	**or**: Is used in expressions
^	YES	**xor**: Is used in expressions
~	YES	**bit-wise negation**: Is used in logic expressions **error return value**: Is used when \W has no WFC chars to return
<		**less than**: Is used in integer and real expressions
>		**greater than**: Is used in integer and real expressions
<=		**less than or equal to**: Is used in integer and real expressions
>=		**greater than or equal to**: Is used in integer and real expressions
==		**equal to**: Is used with pattern data expressions (i.e., to compare strings of WFCs)
!=		**not equal to**: Is used with pattern data expressions (i.e., to compare strings of WFCs)

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

Table 2—Additions to STIL reserved characters *(continued)*

Char	New in STIL.1	Usage
:==	YES	**equal to**: Is used with integer and real expressions (i.e., to compare integer or real numbers)
<>	YES	**not equal to**: Is used with integer and real expressions (i.e., to compare integer or real numbers)
?:		**conditional**: Is used in expressions
=		**assignment**: Is used in signal variable expressions, vector data, group name definitions, spec category expressions, and spec variable expressions
:=	YES	**assignment**: Is used in integer and real expressions
_	YES	**underscore**: Is used as a separator in integer values
.		**dot**: Is used to designate macro and procedure calls in FailData block
:		**colon**: Is used to designate subcomponents of a design (i.e., embedded cores)
::	YES	**double colon**: Is used to designate a domain reference in expressions
{* *}		**brace, asterisk**: Is used to designate a block of data in the case of Pragma (same as delimiters used for annotation blocks)

4.3 Reserved UserFunctions

The user function names in Table 3 are reserved for use in expressions.

Table 3—UserFunctions

Name	New in STIL.1	Usage
min		**function**: Is used in integer or real expressions to return the minimum value from a list of values (as defined in STIL.0).
max		**function**: Is used in integer or real expressions to return the maximum value from a list of values (as defined in STIL.0).
MergedScan	YES	**function**: Is used in patterns to test whether scan in/out data have been merged. See 16.5.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

4.4 Signal and group name characteristics

This clause defines extensions to STIL.0, Subclause 6.10.

All constructs and definitions in STIL.0, Subclause 6.10 are in effect. In addition, signals expressed with a bracketed construct shall allow the use of a previously declared integer-constant, in which integer values are allowed. See 5.1 for the definition of IntegerConstant.

4.5 STIL name spaces and name resolution

Additions to STIL.0, Subclause 6.16.

The Environment block augments the STIL name space as defined in Table 4. This table is incremental to STIL.0, Table 6; all definitions present in that table remain unchanged.

Variable names and constant names are used by expressions. This name space also contains signal names, signal group names, and spec variables defined for the context of that expression. Referenced variable-names shall be unique against all signal names, signal groups names, and spec variable names defined for a PatternBurst or application of a Pattern referenced by that PatternBurst.

Table 4—Additions to STIL name spaces

STIL block	Type of name	Domain restrictions
Environment	Environment domain names	Supports a single unnamed block and domain named blocks. Domain names shall be unique across all Environment blocks.
Variables	Variables domain names	Supports a single unnamed block and domain named blocks. Domain names shall be unique across all Variables blocks.
Variables, Signals, SignalGroups, Spec	Signal names SignalGroup names SignalVariable names Spec variable names Integer names IntegerConstant names WFCConstant names	Names present in this name space are dependent on the domain reference statements in the PatternBurst (SignalGroups domains, Variables domains) and the PatternExec (Category domains). It is an error for defined Variable or Constant names to conflict with Signals, SignalGroups, or Spec variable names accessible in the same context.
ScanStructures	ScanCell names ScanChain names (also used for scan segments and cell groups)	Names present in this space are dependent on the domain reference statements for ScanStructures in the PatternBurst. Scan cell names, scan chain names, and cell group names in this common name space are unique except if an unnamed ScanStructures block exists. Names present in an unnamed block are available unless hidden by a definition of the same name in a named and referenced ScanStructure block.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

5. Expressions

This clause defines extensions to STIL.0, Clause 6.

STIL.0 defines a limited usage of expressions; see STIL.0, Subclause 6.13 and Subclause 6.14. This standard extends these expression capabilities with additional variable types (SignalVariable, Integer, IntegerConstant, WFCConstant) and additional expression constructs. The use of operators and the expression constructs as defined for STIL.0 is unchanged. The full set of STIL.0 and STIL.1 expression operations are defined in Table 7. The detail of syntax and usage of the new expressions are defined in 5.1 – 5.13.

5.1 Constant and variable expressions

Expression constructs (which are defined in 5.2 – 5.13) can be divided into two classes: those that can be completely resolved at parse time (i.e., constant expressions) and those that can only be resolved at run time (i.e., variable expressions).

Constant expressions are those that contain only literal values (e.g., ‘5ns’ or ‘10ns’), or named constants (e.g., Spec-Variable, IntegerConstant, and WFCConstant). As in STIL.0, statements that contain constant expressions can be fully resolved once the STIL file/stream has been fully parsed and the domain references have been resolved. Constant expressions can be used anywhere that the syntax definitions allow a literal value. The use of constant expressions provides for (1) improved readability, (2) parameterized STIL data, (3) enhanced reuse, and (4) improved maintainability.

Variable expressions are those that contain named variables, such as SignalVariable and Integer. The use of variable expressions is intended primarily to support “design” applications. The ability of an ATE system to support variable expressions may be limited, if possible at all.

Both constant and variable expressions can be further classified into “arithmetic expressions” and “pattern data expressions.” The allowed constructs of each of these expression types are defined in 5.2 – 5.13.

5.2 Expression delimiters—single quotes and parentheses

Single quotes are defined in STIL.0 as the delimiter to be used around expressions. This usage is unchanged with the introduction of new expression capabilities in STIL.1, and a STIL.0 file is completely compatible with all STIL.1 rules. However, the rules governing the use of single quotes and parentheses are relaxed. The following rules apply to the use of single quotes in STIL.1:

a) Single quotes, when used, are always the outermost symbols of an expression.

```
'T2+5ns'      // timing expression
'SIG1+SIG2+SIG3'      // signal group expression
```

b) Assignment and boolean operators shall not be used inside a single-quoted expression.

```
V { 'INT := INT+5'; }      // illegal
If 'INT :== 5' {   }      // illegal
```

c) Single quotes shall not be nested.

```
V {'BUS[1..'K+1']' = XXX; }    // illegal - nesting of quotes
```

d) Wherever expressions are allowed, a single token may be used without delimiters. A single token may be a literal value, a named constant, or a named variable.

```
V { INT := '5'; }      // single token with quotes
V { INT := 5; }      // single token without quotes
V { INT1 := INT2; }      // single token without quotes
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: IEC 62526 2007

e) Wherever expressions are allowed, they may be represented without delimiters, with the exception that if a multiterm expression is within a complex data stream, then the delimiters are required. Examples of cases in which delimiters are required are shown in the later examples, using parentheses.

```
V { INT := 'INT+1'; }
V { INT := INT+1; }
```

f) Parentheses may be used inside a single-quoted expression.

```
V { INT := '(INT+1)*2'; }
V { '(GRP-SIG)+SIG' = 00001; }     // make SIG the last WFC in the list
```

Parentheses are the preferred method of delimiting expressions in STIL.1. Parentheses can be used around the entire expression or inside the expression to specify the order of processing. The following rules apply to the use of parentheses:

g) Parentheses are allowed anywhere that single quotes are allowed in STIL.0.

```
V { (SIG1+SIG2) = XX; }
01 { (T2+5ns) D/U; }
```

h) Assignment expressions may be constructed with or without the use of delimeters.

```
V { INT := INT+1; }
V { INT := (INT+1); }
V { (INT := INT+1); }
V { INT := (X+2)*2; }     // parens required to define order
V { SIG1+SIG2 = XX; }
V { (SIG1+SIG2) = XX; }
V { (SIG1+SIG2 = XX); }
```

i) Parentheses are not required to delimit boolean expressions; however, in practice, they are commonly used.

```
If (\W(A+B) == \wxx) { }
If \W(A+B) == \wxx { }
If (\WGRP != \w\r(K+12) X) { }
If \WGRP != \w\r(K+12) X { }
```

j) Parentheses may be nested.

```
V {SIG = \r(K+12) X; }
If (\WBUS[1..(K-1) != \wXXX) { }
```

k) Parentheses are used to delimit parameters to functions within expressions.

```
V { INT := Min(X,Y); }
```

l) If a multiterm expression is within a complex data stream, then delimiters around the expression are required.

```
SIG1 + SIG2 = XX;
\W(SIG1 + SIG2) == XX;     // parens required
AB { 2ns D/U; }
AB { TIM D/U; }
AB { (TIM + 2ns) D/U; }     // parens required
V { GRP = \r2 X YZ; }
V { GRP = \rTWO X YZ; }
V { GRP = \r(TWO+1) X YZ; }     // parens required
V { GRP[5 9] = XX; }
V { GRP[K5 K9] = XX; }
V { GRP[(KK+5) (KK+9)] = XX; }     // parens required
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

5.3 Arithmetic expressions—integer, real, time, boolean

Arithmetic expressions are used in various STIL statements. In the statement syntax definition within this standard, the reference identifiers, *integer_expr, real_expr, boolean_expr,* and *time_expr*, indicate that an arithmetic expression is expected, and they signify the expected output from the expression evaluation. It is allowed to use an arithmetic expression anywhere a literal arithmetic value is allowed as long as it contains only constants and literals. This subclause defines the common rules for all arithmetic expressions.

Arithmetic expressions use the following operators:

— := assignment operator

— ?: conditional operator

— :== compare for equality; return integer 1 if equal; else return 0

— <> compare for inequality; return integer 1 if not equal; else return 0

— < compare less than; return integer 1 if less; else return 0

— > compare greater than; return integer 1 if greater; else return 0

— <= compare less than or equal to; return integer 1 if less or equal; else return 0

— >= compare greater than or equal to; return integer 1 if greater or equal; else return 0

The one exception to the use of := for integer assignments is in the Spec block. For compatibility with STIL.0, it is allowed to use either a bare '=' or the ':=' in this context.

Arithmetic expressions use the following operands:

— Integer, IntegerConstant that are in scope according to the Variables context

— Spec variables that are in scope according to the Category selection

— Timing event labels when used inside a Timing block as defined in STIL.0

— Literal integers, real numbers, and engineering numbers

Pairs of single quotes are used as delimiters in arithmetic expressions according to the rules as defined in STIL.0. It is not allowed to nest single quotes, and the usage should be limited to simple expressions (e.g., 'T2+5ns'). Some statements require that delimiters always be used around expressions, even when the expression is a single literal value or name. If not so stated in the statement definition, then single operands may be used without any delimiters.

Paired parentheses are the preferred method of delimiting expressions in STIL.1. Parentheses can be used around the entire expression or inside the expression to specify the order of execution.

Multiple assignment operators are not allowed in an arithmetic expression:

```
V { INT1 := INT2 := 0; }    // illegal use of multiple assignment operators
```

The following examples are of keyword identified expressions:

```
Loop 50 {}
ScanLength 100;
Loop K+5 {}
Loop 'K+5' {}
Loop (K+5) {}
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: IEC 62526 2007

The following examples are of assignment expressions:

```
Condition { INT := 5; }
V { INT := 5; }
C { INT := 5; }
C { INT := INT + 1; }
C { INT := K; }
C { INT := (INT:==0) ? 99 : INT-1; )
```

The following examples are of boolean expressions:

```
If (INT :== 6) { }
If (INT >= K*2) { }
If (INT > SPEC) { }
If (SPEC < 250ns) { }
While (INT <> 13) { }
```

The following example is of an arithmetic expression within a pattern data expression:

```
V { GRP[1..(K+1)] = \r(K+1) X; }
```

5.4 Pattern data expressions

Whereas arithmetic expressions operate on numeric data, pattern data expressions operate on lists of WFCs. To differentiate the two, a different set of operators is used:

— = assignment operator

— == compare for equality; return integer 1 if equal; else return 0

— != compare for inequality; return integer 1 if not equal; else return 0

— ?: conditional operator

— The pattern data operators that are preceded with a backslash (see Clause 15)

Pattern data expressions use the following operands:

— Signals

— SignalGroups that are in scope according to the PatternBurst context

— Literal lists of WFC's

— SignalVariables and WFCConstants that are in scope according to the Variables context

— Formal parameters from Procedures and Macros

Single quotes are used as delimiters in pattern data expressions according to the rules as defined in STIL.0. It is not allowed to nest single quotes, and the usage should be limited to identifying signal groups (e.g., 'SIGA+SIGB,' 'BUS[1..10]'). Some statements require that delimiters always be used around expressions, even when the expression is a single literal value or name. If not so stated in the statement definition, then single operands may be used without any delimiters.

Unlike arithmetic expressions, parentheses are not to be used on pattern data assignment expression. Assignments should be of the form: TWOSIGS=XX; or 'SIGA+SIGB'=XX;. Parentheses may be used when needed to control order of evaluation as in the following: '(XBUS-XBUS[5])+YBUS'=11110000;.

Multiple assignment operators are not allowed in pattern data expressions:

```
V { SIG1 = SIG2 = X; }      // illegal - use of multiple assignment operators
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

The following examples are of pattern data assignment expressions.

```
V { SIG = X; }      // assign a WFC to a signal
V { GRP = XXXXXX; }      // assign WFCs to a signal group
V { SIGVAR = XXX; }      // assign WFCs to a signal variable
V { GRP[5..6] = XX; }      // selective assign of WFCs
V { 'A+B+C' = XXX; }      // assign WFCs to multiple signals
V { SIGVAR = \W SIG; }      // assign WFC from a signal to a variable
V { SIG = \W SIGVAR[3]; }      // assign WFC from a variable to a signal
V { GRP = \W HALT; }      // assign a WFC-const to a signal group
V { GRP[1..12] = \W SIGVAR[1..4] XXXX \W SIGVAR[5..8]; }
```

The following examples are of pattern data boolean expressions. Note the use of \W on both sides of the expression. The \W operators is required to indicate that WFC data are being compared. The \W operator is used to extract WFC data from a named entity. The \w operator is used to indicate that literal WFC data follow. Note also that when literal WFC data are used, it may appear on either side of the expression.

```
If (\WSIGVAR[6..5] == \wXX) { }
If (\wXX == \WSIGVAR[6..5]) { }
If (\WSIGVAR1 != \WSIGVAR2) { }
```

The following example is of a pattern data expression inside an arithmetic expression:

```
V { INT := (\WGRP==\w010) ? 1 : 0; }
```

The following complete example of statements contains both arithmetic and pattern data expressions:

```
STIL 1.0 { Design 2005; }
Header {
    Source "STD 1450.1-2005";
    Ann {* sub-clause 5.4 *}
}
Signals { S[1..4] In; }
SignalGroups { SIGGRP = 'S[1..4]'; }
Variables {
    IntegerConstant RUN := 0;
    IntegerConstant EXIT := 1;
    IntegerConstant LOAD := 2;
    IntegerConstant UNLOAD := 3;
    Integer CMD;    // use values RUN, EXIT, LOAD, UNLOAD
    Integer INT;
    Integer HEX {InitialValue 0x1000;}
    Integer CC;
    Integer DD;
    SignalVariable SIGVAR1[1..4];
    WFCConstant STOP=00;
    WFCConstant GO=01;
    WFCConstant RESET=10;
    SignalVariable SIGVAR2[1..2];   // use values STOP, GO, RESET
}
Spec {
    Category CAT {
      TIME = '25ns';
    }
}
Pattern PAT {
    C {INT := 1;}
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: IEC 62526 2007

```
31:     If (INT) {} Else {}          // Execute the If block, INT is True
32:     C { INT := 0; }
33:     If (INT) {} Else {}          // Execute the Else block, INT is False
34:     If (INT :== 956) {}          // a boolean expression
35:     If (TIME >= 20ns) {}         // a real number in engr units
36:     If (CC <= DD) {}             // test for CC less than or equal to DD
37:     If (INT < min(CC,DD)) {}     // use of the min function
38:     C {INT := 5;}                // set variable INT equal to 5
39:     C {CMD = \W RUN;}            // set using constant definition
40:     C {HEX := 0xFF;}             // set integer to 255
41:     If (CMD <> LOAD) {}          // integer compare to a constant
42:     C {INT := INT+1;}            // integer expression
43:     C {SIGVAR1 = LH01;}          // set signal variable to the string LH01
44:     V {SIGGRP = \WSIGVAR1;}      // set signals equal to a variable
45:     If (\WSIGVAR2 == \WSTOP) {}  // compare with a constant
46:     If (\WSIGVAR2 == HH) {}      // compare with a WFC list
47: } // end Pattern
```

5.5 Expression processing

This subclause defines the rules for evaluating expressions. Please refer to 5.6 – 5.12 for the definition of the expression types.

Expressions are built up in a hierarchical manner; internal subexpressions are processed, and they, in turn, are used in the next level of the hierarchy. A subexpression is legal expression syntax that is used within another expression. The only distinction between an expression and a subexpression is that a subexpression is a part of another expression.

At the lowest level are the primitive terms (Table 5). A primitive term used by itself represents a legal expression. Each primitive term represents a value and a type. In some cases, the value may not be determinable until run time. In all cases, the type is determinable when the STIL file/stream is parsed.

Subexpressions may be combined with operators to form subexpressions at the next higher level in the expression hierarchy. Like the primitive terms, each subexpression represents a value and a type. And, as with the primitive terms, the value may not be determinable until run time, but the type is determinable when the STIL file/stream is parsed.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

Table 5—Primitive terms

Term	Type	Example—Comment
Integer	numeric	5, 1066
Hex integer	numeric	0xFFFF, 0x123
SI units	numeric	2mV, 25ns —SI units are defined in Table 3 and Table 4 of STIL.0
Real	numeric	2e+3, 4.5e6, 25e-9
Time event reference	numeric	@, @1, @@, @T1, @T1.1 —used in the context of a Timing->Waveform block
Time label	numeric	TXX: —used in the context of a Timing->Waveform block
Signal Signal group	sigref	NAME, “NAME”
Signal Signal group Signal variable WFC constant	WFC list	\WNAME, \W NAME, \W “NAME” —use \W to extract the WFC list fom the named entity
String	WFC list	10HLXT, \w 10HLXT, \h01 FF, \d01 255 —the \ is required with the boolean operators (== <>) —the dot notation for concatenation is neither required or allowed in WFC data; a whitespace character shall serve as separator for WFC lists of greater than 1024 characters
Integer list	integer list	1359 —an ordered set of integers, typically used in the formation of signal groups and scan chains
Bare string	string	STRING, “STRING”, “STRING WITH SPACES”, “STRING WITH SPECIAL CHARS @$%^” —used to reference named entities
Concatenated string	string	XXX.YYY yields XXXYY —used when a string is greater than 1024 characters
Domain reference	string	DOM::NAME —used to reference an item within a named block, i.e., within a named SignalGroups, Timing, or ScanStructures block
Subcomponent reference	string	COMP:NAME, COMP:DOM::NAME —reserved for referencing subcomponents of a design (i.e., embedded cores)
	boolean	—a boolean type is the result of a boolean operation, i.e., ==, :==, ! (see Table 7) —integer 0 is returned for a boolean true result, and integer 1 for a boolean false result —the values TRUE and FALSE are not defined —there is no variable type for boolean

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

Table 6 defines implicit casting rules that are allowed on an expression to change a number from one type of primitive term to another. The casting rule shall be used when necessary to (1) conform to the requirements of the next level of hierarchy, (2) conform to the requirements of the statement containing the expression, or (3) compare a real to an integer number.

Table 6—Implicit numeric casting rules[a]

Term	Cast to	Comment
SI units	real	—used when expression requires a real number
real	integer	—used when expression requires an integer value —the fractional part is truncated; i.e., 3.7 yields 3
integer	real	—used when comparing an integer to a real number
integer	boolean	—used when expression requires a boolean value —value of 1 or greater yields TRUE —value of 0 or negative yields FALSE

[a]If the required behavior is different from the implicit type casting rules, then explicit casting should be performed via user functions.

Table 7 is expanded from Table 5 of STIL.0 and includes the new STIL.1 operators. The behavior of the logic operators is defined in IEEE Std 1364™-2001 (Verilog) [B2]. This table is ordered by precedence. Operators within each double-line separated group are of equal precedence and are processed from left to right. Each double-line separated group of operators has higher precedence than the groups below them and shall be processed first.

Table 7—Operators and functions allowed in expressions

Operator	New	Definition	Number operands	Requirements of operands	Type of the result
Min ( )		minimum value function	>=1	all of same numeric type	same as operands
Max ( )		maximum value function	>=1	all of same numeric type	same as operands
( )		parentheses	1	any	same as operand
[ ]		square brackets	1	integer or integer list	integer list
’		comma - used as an argument separator in functions	2	any	same as operands
!	STIL.1	negation (unary)	1	boolean or sigref	same as operand
~	STIL.1	bit-wise negation (unary)	1	integer	same as operand
+		plus (unary)	1	numeric	same as operand
-		minus (unary)	1	numeric	same as operand
/		divide	2	numeric	numeric, SI units adjusted per rules

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

Table 7—Operators and functions allowed in expressions *(continued)*

Operator	New	Definition	Number operands	Requirements of operands	Type of the result
*		multiply	2	numeric	numeric, SI units adjusted per rules
%	STIL.1	modulus	2	integer	integer
+		add	2	same numeric type	same as operands
-		subtract	2	same numeric type	same as operand
+		concatenate signal list	2	signal or signal group	signal group
-		remove from signal list	2	signal or signal group	signal group
..		integer range (two consecutive dots)	2	integer	integer list
	STIL.1	whitespace	2	integer or integer list	integer list
<	STIL.1	less than	2	same numeric type	boolean
>	STIL.1	greater than	2	same numeric type	boolean
<=	STIL.1	less or equal	2	same numeric type	boolean
>=	STIL.1	greater or equal	2	same numeric type	boolean
= =	STIL.1	equal (WFC)	2	WFC list	boolean
!=	STIL.1	not equal (WFC)	2	WFC list	boolean
:==	STIL.1	equal (numeric)	2	same numeric type	boolean
<>	STIL.1	not equal (numeric)	2	same numeric type	boolean
&	STIL.1	bit-wise and	2	integer or sigref	same as operands
^	STIL.1	bit-wise exlusive or	2	integer or sigref	same as operands
^~, ~^	STIL.1	bit-wise equivalence	2	integer or sigref	same as operands
|	STIL.1	bit-wise inclusive or	2	integer or sigref	same as operands
&&	STIL.1	and	2	boolean	boolean
||	STIL.1	or	2	boolean	boolean
?:		conditional expression	3 (1?2:3)	1: boolean 2,3: same type—numeric or WFC list	same as operands 2, 3
=		assignment (WFC)	2	—left: must be a name of a symbol that can be assigned WFCs —right: a WFC list	WFC list
:=	STIL.1	assignment (numeric)	2	—left: must be a name of a symbol that can be assigned to same type —right: any numeric type	same as second operand

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

5.6 Boolean—*boolean_expr*

An expression is interpreted as a boolean if the context in which it is used requires a boolean result. A boolean result can be achieved by means of the boolean operators. For pattern data expressions, the boolean operator shall be either == or !=, and both sides shall be of type WFC (i.e., a list of WFCs preceded by \w, else a signal, signal group, signal variable, or WFC-constant preceded by \W). For arithmetic expressions, the boolean operator shall be either :==, <> <=, or >=, and both sides shall be of type integer or real.

An expression that results in an integer or real value shall be interpreted as a boolean if the usage context so requires; in which case, a nonzero positive value is interpreted as true, whereas a value of zero or negative is interpreted as false. Expressions that evaluate to WFCs shall not be used as booleans.

Although it may be good practice to use parentheses around boolean expressions, this is not a requirement of the language.

The following examples are of boolean expressions:

```
STIL 1.0 { Design 2005; }
Header {
    Source "STD 1450.1-2005";
    Ann {* sub-clause 5.6 *}
}

Variables {
    Integer INT;
    Integer FLAG;
    IntegerConstant FALSE := 0; // FLAG value
    IntegerConstant TRUE := 1; // FLAG value
    SignalVariable OP_CODE[3..0];
    WFCConstant RUN = 1011; // OP_CODE value
    WFCConstant STOP = 0111; // OP_CODE value
    WFCConstant RESET = 0000; // OP_CODE value
} // end Variables

Pattern MY_PAT {
    If (INT :== 13) {}
    If (FLAG :== TRUE) {}
    If (\WOP_CODE == \WRESET) {}
    If (\w1111 == \WOP_CODE) {}
} // end Pattern
```

5.7 Integers—*integer_expr*

An integer expression is an expression that evaluates to integer. Integer expressions are allowed anywhere that the statement syntax calls for an "*integer_expr*". See the list of allowed operators in Table 7. The following rules of interpretation apply to integers and integer expressions:

a) A bare integer may be declared either with or without single quotes, e.g., 5 or '5'.

b) The underscore character may be used as a separator within an integer declaration, e.g., 65_535.

c) If an integer result is called for and the expression results in a number with a fractional part, the fraction is truncated to produce an integer value; e.g., the values [6.1, 6.8, –3.1, –3.8] become [6, 6, –3, –3], respectively. The truncation is performed at the conclusion of the evaluation; i.e., (3/2) × 2 results in a value of 3.

d) The following formats are not allowed for integers: 56E3, 56K.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

e) Integer may be expressed in hexadecimal by preceding the number with 0x, e.g., 0xFF.

Integer expressions may comprise integer literals, integer variables, integer constants, or multiterm expressions that result in an integer value. The following examples are of integer expressions and their usage:

```
STIL 1.0 { Design 2005; }
Header {
    Source "STD 1450.1-2005";
    Ann {* sub-clause 5.7 *}
}

Signals { SIGA In; SIGB Out; SIGC InOut; FOO[0x0..0x3F] In; }
Variables {
    Integer II;
    Integer JJ { InitialValue 56*1024; }
    IntegerConstant KK := 2;
}
SignalGroups {
    ALL = 'SIGA+SIGB+SIGC+FOO';
}   // end SignalGroups

Timing { WaveformTable WFT {
    Waveforms {
      ALL {
        01 { '2ns+KK*0.5ns' U/D; }
      }
    }   // end Waveforms
}}   // end Timing

Pattern P {
    C { II := 1234; }
    C { II := 48000000; }
    C { II := 48_000_000; }        // equivalent to 48000000
    C { II := II + 2; }            // expression with a variable
    C { II := 13; JJ := 13; }      // expression containing assignment
    C { II := JJ/2; }              // truncate (i.e., 13/2 yields 6)
    C { II := JJ*0.5; }            // truncate (i.e., 13*0.5 yields 6)
    C { II := 0x7FF_FFFF; }        // hex representation of integer
    C { II := JJ&0x00ff; }         // bitwise and with a hex number
    If (II >= 99) {}               // conditional expression
    Loop II {}                     // variable loop count
    Loop 0xff { }                  // loop count defined in hex
} // end Pattern
```

5.8 Logic expressions—*logic_expr*

A logic expression is used to represent a combination of signal names and signal group names in the context of a design model. Logic expressions are allowed anywhere that the statement syntax calls for a "*logic_expr*". The rules for interpretation of this expression are outside of the definition of this standard (see IEE Std 1364-2001 [B2]).

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

The logic relationship represented by this expression is independent of any timing or waveform considerations. It is solely indicating that the intended function of the design is to combine the named signals and signal groups according to the specified logic relationship.

The allowed operators in a logic expression are as follows:

( )	parentheses
~	bit-wise negation
&	bit-wise and
|	bit-wise inclusive or
^	bit-wise exclusive or
^~, ~^	bit-wise equivalence

The following example contains a logic expression in the ScanEnable statement. The scan enable condition for the scan chain CHAIN1 is active if signals AA and BB are both true, thereby satisfying the logic expression 'AA&BB'.

```
STIL 1.0 { Design 2005; }
Header {
   Source "STD 1450.1-2005";
   Ann {* sub-clause 5.8 *}
}

Signals { AA In; BB In; SI1 In; SO1 Out;}

ScanStructures S1 {
   ScanChain CHAIN1 {
     ScanLength 100;
     ScanCells CC[1..100];
     ScanIn SI1;
     ScanOut SO1;
     ScanEnable AA&BB;
   }
}
```

5.9 Real expressions—*real_expr*

A real expression is an expression that evaluates to a real number as determined by the context (i.e., key word) or the variable type on the left-hand side of an expression. Real expressions are allowed anywhere that the statement syntax calls for a "*time_expr*" or "*real_expr*"; i.e., anywhere that an engineering unit value is allowed, then a real expression is allowed. Real variables and expressions are defined in a Spec block in STIL.0. A real number can be expressed in one of two formats as follows:

a) A real number can be of the form: <number>e<+\-><number>. A real number can be used to represent values that are not standard SI units, for example, a slew rate in volts/nanoseconds.

b) A real number can also be of the form <number><prefix><SI unit>. For example, '23ns' is a time expression (nanoseconds), '10uF' is a capacitance expression (micro-Farads). This generic reference can be used whenever one of the standard unit definitions is allowed.

There are commonly accepted rules with regard to the algebraic combination of values with engineering units. Such rules are not defined in this standard; however, specific tools that implement this standard may require compliance with these rules, and it is good practice to keep units consistent with the usage. For example, if a spec value defines VX to be of type voltage, it can be used as time by expressing it as VX*(1ns/1V).

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

The following examples are of real number expressions:

```
STIL 1.0 { Design 2005; }
Header {
   Source "STD 1450.1-2005";
   Ann {* sub-clause 5.9 *}
}

Signals { SIG_NAME InOut; }

Spec {
   Category CAT {
     TIME = 25ns;
     REALVAR = 5.5;
     VOLTAGE = 2.5V;
     WATTAGE = 25mW;
     SLEWRATE = 1V/1ns;
     } // end Category
} // end Spec

Pattern PAT {
   If (TIME >= 23.5ns+1.5ns/2) {}            // time expression
   If (WATTAGE >= 5V*2uA) {}                 // where WATTAGE is of type 'Watts'
   If (SLEWRATE' >= 5V/1ns) {}               // where SLEWRATE is of type 'Real'
   If (REALVAR >= 5) {}                      // where REALVAR is of type 'Real'
   If (REALVAR*1mA >= 5mA) {}                // where REALVAR is of type 'Real'
} // end Pattern

Timing {
    WaveformTable WFT {
      Waveforms { SIG_NAME {
        01 { 25ns D/U; }                     // constant time
        01 { (REALVAR*1ns) D/U; }            // use a real to define time
        01 { (VOLTAGE*(1ns/1V)) D/U; }       // use a voltage to define time
      }}
    }
} // end Timing
```

5.10 Addition to timing expressions—*time_expr*

Additional symbols are added for referencing timing events in waveforms. These are additions to the @n symbol (as defined in STIL.0, Subclause 6.13), which allows reference to the timing edges in a timing expression that are in the current period. The new symbols, as defined here (along with the other time referencing capabilities), allow access to period markers and events in subsequent periods. These new facilities are provided to support application in other STIL extensions, such as STIL.3 and STIL.6.

a) @ => references the time of the prior event (STIL.0).

b) @n => references the time of the n-th event, where first event is @1 (STIL.0).

c) @@ => references the time of the current event (STIL.1).

d) @Tm => references the time of the start of the m-th period, where the current period is numbered 0. The time value returned is relative to T0 of the current period. @T1 represents the end of the current period (i.e., the time relative to the current period for the start of the next period).

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

e) @Tm.n => references the events of subsequent periods. The time value returned is relative to T0 of the current period. @T1.3 represents the third event of the second period. Note that events are numbered starting at 1, because the 0-th event is T0 (STIL.1).

f) EVENT_LABEL => references a label that is defined by EVENT_LABEL: anywhere in the current WaveformTable or WaveformDescriptions block (STIL.0).

g) SPEC_VARIABLE => references a variable defined in a Spec block (STIL.0).

h) VARIABLE => references an Integer or IntegerConstant defined in a Variables block (STIL.1).

Figure 1 illustrates events across three periods.

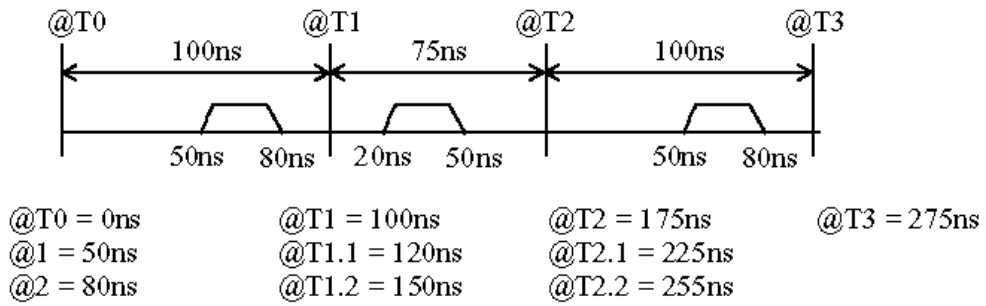

Figure 1—Referencing timing edges

5.11 SignalVariables—*sigvar_expr*

A SignalVariable is a variable that is used to hold strings of WFC characters. It can be assigned WFC strings in the same way that signals and signal groups are assigned WFC strings. A *sigvar_expr* is a reference to a signal variable, with the result being a list of WFCs. A SignalVariable is assigned a list of WFC values, which may then be assigned to actual signals or groups. The list of WFC values contained in a variable of type SignalVariable is obtained by preceding it with \W. A SignalVariable name may be used to pass WFC information as a parameter in a macro or a procedure calls.

SignalVariables can be used in-line by setting them to a list of WFCs in a vector or a condition statement. When used in this way, the data are immediately available for use. Note: See also the definition of formal parameters in the next section. The following example shows in-line use of a signal variable:

```
C { SV = 1111; }
V { GRP = \WSV; }
```

The contents of the SignalVariable are maintained on a call to a procedure and the return; and the content is maintained on the invocation and exit from a macro.

The WFC data in a SignalVariable can be used in much the same way as Signals and SignalGroups. Square brackets are used to indicate the length if there are multiple elements. The following example illustrates typical SignalVariable definitions:

```
Variables {
  SignalVariable SV1;      // define a single element variable
  SignalVariable SV3[0..2];      // define a three element variable
}
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: IEC 62526 2007

When used in pattern data, the SignalVariable may be used without the square brackets; in which case, all elements of the SignalVariable are used. If a subset of the elements is desired, then the square bracket notation is used. The length on both sides of the equal sign must be the same, except when the signal-variable is being used as a formal parameter (see 5.12). The following example illustrates typical SignalVariable usage:

```
Pattern
  V { SIG = \W SV1; }
  V { GRP_OF_THREE = \W SV3; }
  V { GRP_OF_TWO = \W SV3[0..1]; }
}
```

The following examples are of SignalVariable usage:

```
STIL 1.0 { Design 2005; }
Header {
   Source "STD 1450.1-2005";
   Ann {* sub-clause 5.11 *}
}

Signals {
   BUSX[1..5] In;     // defines 5 signals plus a group named BUSX
}

Variables {
   SignalVariable SIG_VAR[1..5];
}

MacroDefs {
   APPLY_VAR {
     C { SIG_VAR[1..5] = #; }
     V { BUSX = \W SIG_VAR[1..5]; }
     // error if above two lines were changed to:
//   V { SIG_VAR[1..5] = #; BUSX = \W SIG_VAR[1..5]; }
   }
   APPLY_TWO_VARS {
     C { SIG_VAR[1..5] = #; }
     V { BUSX = \W SIG_VAR[1..5]; } // apply 11111
     C { SIG_VAR[1..5] = #; }
     V { BUSX = \W SIG_VAR[1..5]; } // apply 00000
   }
}

Pattern PAT {
   // following macro call use signal variables as parameters
   Macro APPLY_VAR { SIG_VAR[1..5] = 11100; }
   Macro APPLY_VAR { SIG_VAR[5 4 2 3 1] = 00011; }
   Macro APPLY_VAR { SIG_VAR[5..1] = ABBAB; }
   Macro APPLY_TWO_VARS { SIG_VAR[5..1] = 11111 00000; }
   // following use signal variables in line
   C { SIG_VAR[1..5] = 11001; }
   V { BUSX = \W SIG_VAR[1..5]; }
}
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub
Full document details: IEC 62526 2007

5.12 Formal parameters in procedures and macros

The concept of a formal parameter was used in STIL.0; however, it was not called that. Now with the introduction of more complicated pattern data expressions, it is necessary to keep the concept of variable usage and formal parameter separate. Whereas a signal variable may be set to a value that becomes immediately available, a parameter in a macro or procedure is only available when it is referenced within the referenced macro or procedure by the use of the # or % characters. It is the same behavior as for any Signal or SignalGroup that is used as a parameter as defined in STIL.0. See the following example:

When the name of a SignalVariable is used as a parameter in a macro or a procedure call:

```
Macro MAC { SV = 1111; }
Call PROC { SV = 1111  0101; } )
```

then the WFC data are not available until a # or % operator is used inside the procedure or macro:

```
MAC {
  C { SV=#; } V { GRP=\WSV; }
}
PROC (
  C { SV=#; } V { GRP=\WSV; }
  C { SV=#; } V { GRP=\WSV; }
}
```

5.13 Integer lists—*integer_list*

An *integer_list* is used to specify an ordered list of integer values. Integer lists are allowed anywhere that the statement syntax calls for an "*integer_list*." The only allowed operator in an integer list is the integer range operator "..", which specifies a range of integers. An *integer_list* may contain either single integers, whitespace-separated lists of integers, or integer ranges. IntegerConstants may appear in place of integers. An integer range is represented by two integers (or IntegerConstants) separated by "..". For example, '3..6' is equivalent to '3 4 5 6' and '4..2' is equivalent to '4 3 2'. Whitespace characters are allowed between the integer or IntegerConstant, and the ".." operator. The following code shows examples of integer lists:

```
200: STIL 1.0 { Design 2005; }
201: Header {
202:   Source "STD 1450.1-2005";
203:   Ann {* sub-clause 5.13 *}
204: }
205: Signals {
206:   SIG[1..5] In;
207:   SI1 In { ScanIn 100; }
208: }
209: ScanStructures S1 {
210:   ScanChain CHAIN1 {
211:     ScanLength 100;
212:     ScanCells CC[1..100];
213:   }
214: }
215: Pattern PAT {
216:   V { SIG[5 4 3 2 1] = 11001; }
217:   V { SIG[5..1] = 00110; }
218: }
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

6. Statement structure and organization of STIL information

This standard defines additional top-level STIL blocks: Environment. It is delineated in Table 8, which is provided to be complete with Table 8 in STIL.0.

Table 8—Additions to optional top-level statements

Statement	Purpose
Environment Clause 18	The Environment block defines relationships of STIL data to external environments. Environment blocks, if referencing other STIL data, shall be defined after the blocks that define that STIL data, unless those references are to information contained in MacroDefs, Procedures, or Pattern blocks, which are allowed to be forward-references. This is always a top-level block. Environment blocks, if present, do not have a defined order with respect to other STIL blocks defined in STIL.0. Environment blocks may appear any place a top-level STIL statement is allowed.
Pragma Clause 19	The Pragma block allows for embedding application-specific code directly into a STIL file/stream. The block may be used as a top-level block or embedded in another STIL block.
PatternFailReport Clause 20	The PatternFailReport block is typically not part of a STIL file/stream that defines a test program; however, as a top-level block, it would be allowed to exist without conflict with other STIL blocks. This block is to be generated by the execution of a STIL test program to report results of execution.

7. STIL statement

This clause defines extensions to STIL.0, Clause 8.

The STIL statement identifies the primary version of STIL.0 information contained in a STIL file and the presence of one or more standard Extension constructs. The primary version of STIL is defined in STIL.0.

The extension to the STIL statement allows for a block containing extension identifiers that allow for additional constructs in the STIL file. There may be multiple Extension statements present, to identify the presence of multiple extension environments. The extension name and the extension statements are defined in the individual documents for those standards.

“Include” files are required (per STIL.0) to start with the STIL statement. The extension context as defined in an “Include” file shall be a subset of the definition in the base file; i.e., it is not permissible to specify an extension in an included file that is not already allowed by the base file. One qualifier to this rule is the "IfNeed" option. If the parser does not actually consume the "Include," then the extensions in the file are not relevant.

All other constructs and restrictions for STIL.0, Clause 8 are in effect here.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

7.1 STIL syntax

STIL IEEE_1450_0_IDENTIFIER **{** (1)
(EXT_NAME EXT_VERSION;)+ (2)
} *// end STIL*

(1) **STIL:** A statement at the beginning of each STIL file.

IEEE_1450_0_IDENTIFIER: The primary version of STIL, as identified by 1.0.

(2) EXT_NAME: The name of the Extension. This standard is identified by the name **Design**.

EXT_VERSION: The version of the extension. This standard is identified by the value **2005**.

7.2 STIL example

```
STIL 1.0 {
   Design 2005;
   DCLevels 2002;
}
Header {
   Source "STD 1450.1-2005";
   Ann {* sub-clause 7.2 *}
}

```

8. UserKeywords statement

This clause defines extensions to STIL.0, Clause 11.

This clause defines additional locations in a STIL file/stream where the UserKeywords statement is allowed to appear.

The UserKeywords construct is expanded from STIL.0 to allow the UserKeywords statement to be defined within any STIL block. When a UserKeywords statement is defined within a STIL block, those definitions shall apply only within that block and contained sub-blocks. It allows Userkeywords to be “locally scoped” to a containing STIL block. Note: STIL.0 allows for UserKeywords only at the top level.

All other constructs and requirements for STIL.0, Clause 11 are in effect here.

8.1 UserKeywords syntax

As defined in STIL.0.

8.2 UserKeywords example

```
STIL 1.0 { Design 2005; }
Header {
   Source "STD 1450.1-2005";
   Ann {* sub-clause 8.2 *}
}

Signals { A[1..99] InOut; }
SignalGroups { ALLSIGNALS = 'A[1..99]'; }
```

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

```
236:
237:     /* The UserKeywords construct is the same as defined in STIL.0,
238:     with the additional capability to be defined within another STIL block */
239:
240: Timing ONE {
241:    UserKeywords STARTUP SHUTDOWN;
242:    WaveformTable ONE {
243:      STARTUP { ALLSIGNALS;}
244:    }
245: }
```

9. Variables block

This clause defines variables and named constants. If the block is unnamed, then all definitions shall be globally available to all other blocks in the STIL file/stream. If the block has a name, then that name must be referenced by the Pattern, PatternBurst, or Timing block for the variables to be available. Variables and constants that are defined in the global Variables block shall not be overridden in a named Variables block.

This block should be considered in light of the Spec block that also can define constants. The purpose and usage of these two blocks is sufficiently different to warrant their separation. The Spec block is intended to contain parametric data used in specifying device/test characteristics. The Spec block has special handling defined for Category, Min/Typ/Max/Meas, and Selector, which are to facilitate a flexible definition of these test parameters.

The Variables block is used to contain control variables and constants. Typical uses for these constants and variables is to control flow of execution, to provide 'aliasing' of values to meaningful names with constants, and to provide run-time parameters.

Although complex expressions may be supported by an ATE system, in general, most ATE systems will be able to support only a limited subset of the design capabilities defined herein. The full capabilities are intended for use by simulation applications or pattern translation tools. See Annex O for more information.

9.1 Variables block syntax

Variables (VARIABLES_DOMAIN) { (1)
  (**IntegerConstant** CONST_NAME **:=** INTEGER **;**)* (2)
  (**Integer** VAR_NAME;)* (3)
  (**Integer** VAR_NAME {
    (**Usage Test**;)
    (**InitialValue** *integer_expr* ;)
  })* *// end Integer*
  (**SignalVariable** VAR_NAME;)* (4)
  (**SignalVariable** VAR_NAME {
    (**Base** < **Hex** | **Dec**> WFC_LIST ;)
    (**Alignment** < **MSB** | **LSB** > ;)
    (**InitialValue** *vec_data* ;)
  })* *// end SignalVariable*
  (**WFCConstant** CONST_NAME **=** WFC_LIST **;**)* (5)
} *// end Variables*

(1) **Variables**: This block contains the definitions of variables and constants. It may be either named or unnamed. If the block is named, then the name must be referenced in the PatternBurst block or Timing Block for the variables to be available.

(2) **IntegerConstant**: This statement allows the definition of a named integer that has a constant value. The "INTEGER" value on the right-hand side may be either a literal integer value (as defined in

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

STIL.0, Subclause 6.12) or an integer-expr containing only literal integers and other integer-constant names. The named constant shall be used anywhere that a literal integer value is allowed. An integer-constant can be more widely used than an integer-variable; for example, if K is an integer-constant, then it may be used in defining an indexed list of signals (SIGS[1..K] In;), as a length indicator (ScanIn K; DataBitCount K; ScanOutLength K;), and as a loop counter (Loop K { } MatchLoop K { }).

(3) **Integer**: This statement or block defines an integer-variable. If the integer-variable is specifed as a block, then the following optional attributes may be specified:

Usage Test: This statement specifies that the integer-variable is to be used in the translation, load, or execution of the STIL file/stream for an ATE system. Integer-variables without this statement may safely be ignored for the purposes of pattern execution. The default is NO; i.e., do not use as a test variable.

InitialValue: This attribute allows the specification of the initial value that is to be assigned to the integer-variable at the onset of highest level domain for the variable, i.e., global = upon creation; PatternBurst = each execution of the burst; Pattern = each execution of the pattern. The value of the expression defined in the InitialValue statement is not established until a context is established in which this integer-variable is in scope. The default initial value is zero.

(4) **SignalVariable**: This statement or block defines a variable that may be used wherever a signal or group may be used. For example code, see Annex B.

When declaring signal-variables, either the single WFC entity form may be specified that requires no square brackets or the multiple WFC entity form may be specified with the square bracketed notation. The square brackets accomplish two purposes: defining the length of the signal-variable and defining the index range of the signal-variable.

When using a signal-variable within pattern data, it is an error to access a bracketed index that is outside the declared range of a bracketed signal-variable. A signal-variable consumes one WFC for each index when assigned a WFC list. When declared as a bracketed signal-variable, a reference to that signal-variable without brackets is equivalent to a reference of that signal-variable across the declared bracketed range of that signal-variable.

If the SignalVariable is a block, then the following optional attributes may be specified:

Base <Hex | Dec> WFC_LIST: This attribute defines the mapping to WFC characters from hex or decimal values, as defined for SignalGroups in STIL.0, Clause 14. If no base is specified, the default is to use waveform characters.

Alignment: This attribute is used to specify whether the mapping from hexadecimal to the signal-variable is to align with the MSB or the LSB. The default is MSB to accommodate the convention used for scan data. This attribute applies only to hexadecimal values because decimal values are always LSB aligned.

InitialValue: This attribute allows the specification of the initial value that is assigned to the signal-variable at the onset of each execution of the first or highest-level PatternBurst that uses this signal-variable. Signal-variables declared with no InitialValue are undefined until assigned a value (written to), and it is an error to reference the signal-variable while in the undefined state. The value of the expression defined on the InitialValue statement is not established until a context in which this signal-variable is in scope is initiated. This context occurs under a PatternBurst that references the Variables block that contains this signal-variable, in combination with the PatternExec that invoked this PatternBurst.

(5) **WFCConstant**: This statement allows the definition of an named list of waveform characters that has a constant value. The named constant shall be used anywhere that a WFC or WFC_list is allowed.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007

9.2 Variables example

```
STIL 1.0 { Design 2005; }
   Header {
     Source "STD 1450.1-2005";
     Ann {* sub-clause 9.2 *}
   }
Variables {
     IntegerConstant BUSTOP := 15;
     Integer H1;
     Integer H2 { Usage Test; InitialValue 13; }
     IntegerConstant RED:=0;
     IntegerConstant GREEN:=1;
     IntegerConstant BLUE:=2;
     Integer COLORS;   // Values = RED, GREEN, BLUE
     SignalVariable VW;    // a single WFC variable
     SignalVariable VX[1..5]{ InitialValue 11000; }
     SignalVariable VY[7..1]{ Base Hex AB; InitialValue FE; }
     WFCConstant RESET=00;
     WFCConstant RUN=01;
     WFCConstant EXTEST=10;
     WFCConstant INTEST=11;
     SignalVariable VZ[0..1];   // Values = RESET, RUN, EXTEST, INTEST
     SignalVariable FOO[1..6];
}
Signals {
     BUS1[ BUSTOP .. 0 ] Inout;
     SIG1 In;
     SIG2 In;
     SIG[2..6] In;
   }
SignalGroups {
   GRP = 'SIG[2..6]';
}
Procedures {
   PROC {
     C { FOO=#; VW=#; VX=11111; VY=\h82; VZ=#; }    // VY = BAAA AAB
     V { SIG1=\W FOO[1]; GRP=\W FOO[2..6]; }
     V { SIG2=\WVW; SIG[5..6]=\WVZ; }
   }
}
Pattern P {
   Call PROC { FOO=111000; VW=P; VZ=\WRESET; }
   Call PROC { FOO=010101; VW=P; VZ=\WRUN; }
}
```

9.3 Variables scoping

Care should be taken when defining Variables blocks as to whether the variables and constants are to be shared or unique to a pattern. Shared (global) variables are useful for passing data, whereas nonshared (local) variables provide independent operation. The scope of a variable or constant is determined by (1) whether the Variables block is named or unnamed and (2) how a named Variables block is referenced.

Published by IEC under licence from IEEE. © 2005 IEEE. All rights reserved.

Documents
Document Sharing Hub

Full document details: IEC 62526 2007