

IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data

IEEE Computer Society

Developed by the
Test Technology Standards Committee

IEEE Std 1450™-2023



STANDARDS

IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data

Developed by the

Test Technology Standards Committee
of the
IEEE Computer Society

Approved 6 December 2023

IEEE SA Standards Board

This is a Redline document produced by Accuris.

Redlines are designed to save time and improve productivity by using the latest software technology to find and highlight document changes from one edition to the next. It includes a full-text version of the new document, plus an indication of changes from the previous version.

Because it may not be technically possible to capture all changes accurately, it is recommended that users consult previous editions as appropriate. In all cases, only the current base version of this publication is to be considered the official document. This Redline is produced solely by Accuris. The publisher has not prepared this Redline and is not responsible for its accuracy.

Contact:

<https://store accuristech.com/pages/contact.html>

Redline Processing Notes:

1. ~~Red Text~~ - Red strikethrough text denotes deletions.
2. Blue Text - Blue underlined text denotes modifications and additions.

Abstract: Standard test interface language (STIL) provides an interface between digital test generation tools and test equipment. A test description language is defined in this standard that: (a) facilitates the transfer of digital test vector data from CAE to ATE environments; (b) specifies pattern, format, and timing information sufficient to define the application of digital test vectors to a DUT; and (c) supports the volume of test vector data generated from structured tests.

Keywords: automatic test pattern generator, ATPG, BIST, built-in self-test, CAE, computer-aided engineering, cyclize, device under test, digital test vectors, DUT, event, functional vectors, IEEE 1450™, pattern, scan vectors, signal, structural vectors, timed event, waveform, waveshape

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

Copyright © 2024 by The Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 24 April 2024. Printed in the United States of America.

IEEE is a registered trademark in the U.S. Patent & Trademark Office, owned by The Institute of Electrical and Electronics Engineers, Incorporated.

PDF: ISBN 979-8-8557-0629-1 STD26865
Print: ISBN 979-8-8557-0630-7 STDPD26865

IEEE prohibits discrimination, harassment, and bullying.

For more information, visit <https://www.ieee.org/about/corporate/governance/p9-26.html>.

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

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (<https://standards.ieee.org/ipr/disclaimers>), appear in all standards and may be found under the heading “Important Notices and Disclaimers Concerning IEEE Standards Documents.”

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

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers are not necessarily members of IEEE or IEEE SA 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 makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning this standard, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. IEEE Standards documents do not guarantee safety, security, health, or environmental protection, or guarantee against interference with or from other devices or networks. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. 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 their 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: THE NEED TO PROCURE 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 balloting 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 is 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, nor 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 the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s).

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, 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 subcommittees of the IEEE SA Board of Governors 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 evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE working group. You can indicate interest in a working group using the Interests tab in the Manage Profile and Interests area of the [IEEE SA myProject system](#).¹ An IEEE Account is needed to access the application.

Comments on standards should be submitted using the [Contact Us](#) form.²

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 constitute 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.

¹ Available at: <https://development.standards.ieee.org/myproject-web/public/view.html>.

² Available at: <https://standards.ieee.org/thank-you/contact/>.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US 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, neither IEEE nor its licensors waive any rights in copyright to the documents.

Photocopies

Subject to payment of the appropriate licensing fees, 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; <https://www.copyright.com/>. 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 10 years. When a document is more than 10 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](#) or [contact IEEE](#).³ For more information about the IEEE SA or IEEE's standards development process, visit the IEEE SA Website.

Errata

Errata, if any, for all IEEE standards can be accessed on the [IEEE SA Website](#).⁴ Search for standard number and year of approval to access the web page of the published standard. Errata links are located under the Additional Resources Details section. Errata are also available in [IEEE Xplore](#). Users are encouraged to periodically check for errata.

³ Available at: <https://standards.ieee.org/about/contact>.

⁴ Available at: <https://standards.ieee.org/standard/>

Patents

IEEE standards are developed in compliance with the [IEEE SA Patent Policy](#).⁵

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 via an Accepted Letter of Assurance, then the statement is listed on the IEEE SA Website at <https://standards.ieee.org/about/sasb/patcom/patents>. 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.

IMPORTANT NOTICE

Technologies, application of technologies, and recommended procedures in various industries evolve over time. The IEEE standards development process allows participants to review developments in industries, technologies, and practices, and to determine what, if any, updates should be made to the IEEE standard. During this evolution, the technologies and recommendations in IEEE standards may be implemented in ways not foreseen during the standard's development. IEEE standards development activities consider research and information presented to the standards development group in developing any safety recommendations. Other information about safety practices, changes in technology or technology implementation, or impact by peripheral systems also may be pertinent to safety considerations during implementation of the standard. 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.

⁵ Available at: <https://standards.ieee.org/about/sasb/patcom/materials>.

Participants

At the time this standard was completed, the STIL Working Group had the following membership:

Ric Dokken, *Chair*
Jim O'Reilly, *Vice Chair*
Michael Patnode, *Secretary*

Boon Chong Ang
Dima Angert
Peter Bauwens
Steve Cannon
Julia Dichiaro

Ajay Dimri
Dwayne Dohmann
Dane Knox
David Leslie

Adam W. Ley
John Morris
Paul Reuter
Ramesh Sharma
Bambang Suparjo

The following members of the individual Standards Association balloting group voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Boon Chong Ang
Peter Bauwens
Paul Berndt
Bill Brown
Zhiman Chen
Jonathan Colburn
Ric Dokken
Heiko Ehrenberg

Ian Harrison
Werner Hoelzl
Piotr Karocki
Adam W. Ley
Teresa Lopes
Greg Luri
Gregory Maston
Ian McIntosh

Jim O'Reilly
Bansi Patel
Paul Reuter
Douglas Sprague
Bambang Suparjo
David Thompson
Dmitri Varsanofiev
Oren Yuen

The Working Group thanks those companies that contributed concepts and ideas to this effort, in addition to people and time. These contributions helped to define the language presented in this standard. In particular, the Working Group would like to thank: LTX Corporation, for providing information about the enVision environment; Motorola, Incorporated, for providing information about the Universal Test Interface Code (UTIC); Test Systems Strategies, Incorporated, for providing information about the Waveform Generation Language (WGL); and Texas Instruments, Incorporated, for providing information about the Test Description Language (TDL).

When the IEEE SA Standards Board approved this standard on 6 December 2023, it had the following membership:

David J. Law, *Chair*
Ted Burse, *Vice Chair*
Gary Hoffman, *Past Chair*
Konstantinos Karachalios, *Secretary*

Sara R. Biyabani
Doug Edwards
Ramy Ahmed Fathy
Guido R. Hiertz
Yousef Kimiagar
Joseph L. Koepfinger*
Thomas Koshy
John D. Kulick

Joseph S. Levy
Howard Li
Gui Lin
Johnny Daozhuang Lin
Xiaohui Liu
Kevin W. Lu
Daleep C. Mohla
Andrew Myles

Paul Nikolich
Annette D. Reilly
Robby Robson
Lei Wang
F. Keith Waters
Karl Weber
Philip B. Winston
Don Wright

*Member Emeritus

Introduction

This introduction is not part of IEEE Std 1450-2023, IEEE Standard for Test Interface Language (STIL) for Digital Test Vector Data.

Standard test interface language (STIL) was initially developed by an ad-hoc consortium of test equipment vendors, computer-aided engineering (CAE) and computer-aided design (CAD) vendors, 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 need for a common interchange format for large volumes of digital test data was identified as an overriding priority for the work; as such, the scope of the work was constrained to those aspects of the test environment that contribute significantly to the volume issue, or are necessary to support the comprehension of that data. Binary representations of data were a key consideration in these efforts, resulting in a proposal to incorporate the compression of files as part of this standard.

Limiting the scope of any standards project is a difficult thing to do, especially for a room full of engineers. However, issues that did not impact the scope as identified were dropped from consideration in this version of the standard. The capabilities that are not intended to be part of this first standard are covered in 1.1.

Early work in this consortium consisted of identifying the requirements necessary to address this problem and reviewing existing options and languages in the industry. All options proposed fell short of addressing the requirements, and the consortium started to define a new language. This work was executed with heavy leverage from some existing languages and environments, and STIL owes much to the groundwork established by these other languages.

Contents

1. Overview	15
1.1 Scope	16
1.2 Purpose	17
1.3 Word usage	18
2. Normative references	18
3. Definitions, acronyms, and abbreviations	19
3.1 Definitions	19
3.2 Acronyms and abbreviations	21
4. Structure of this standard	22
5. STIL orientation and capabilities tutorial	22
5.1 General	22
5.2 Hello tester	22
5.3 Basic LS245	27
5.4 STIL timing expressions/"Spec" information	31
5.5 Structural test (scan)	34
5.6 Advanced scan	37
5.7 IEEE Std 1149.1™-1990	43
5.8 Multiple data elements per test cycle	47
5.9 Pattern reuse/direct access test	50
5.10 Event data/non-cyclized STIL information	53
6. STIL syntax description	62
6.1 General	62
6.2 Case sensitivity	62
6.3 Whitespace	62
6.4 Reserved words	62
6.5 Reserved characters	63
6.6 Comments	65
6.7 Token length	65
6.8 Character strings	65
6.9 User-defined name characteristics	65
6.10 Domain names	66
6.11 Signal and group name characteristics	67
6.12 Timing name constructs	67
6.13 Number characteristics	67
6.14 Timing expressions and units (<i>time_expr</i>)	68
6.15 Signal expressions (<i>sigref_expr</i>)	70
6.16 WaveformChar characteristics	71
6.17 STIL name spaces and name resolution	72
7. Statement structure and organization of STIL information	73
7.1 Top-level statements and required ordering	74
7.2 Optional top-level statements	76
7.3 STIL files	76
8. STIL statement	76
8.1 General	76

8.2 STIL syntax	77
8.3 STIL example	77
9. Header block.....	77
9.1 General	77
9.2 Header block syntax	77
9.3 Header example	78
10. Include statement.....	78
10.1 General	78
10.2 Include statement syntax.....	78
10.3 Include example.....	79
10.4 File path resolution with absolute path notation	79
10.5 File path resolution with relative path notation	79
11. UserKeyWords statement.....	79
11.1 General	79
11.2 User Keywords statements syntax	80
11.3 User Keywords example.....	80
12. UserFunctions statement.....	80
12.1 General	80
12.2 UserFunctions statement syntax	80
12.3 UserFunctions example	81
13. Ann statement.....	81
13.1 General	81
13.2 Annotations statement syntax	81
13.3 Annotations example	81
14. Signals block	81
14.1 General	81
14.2 Signals block syntax	82
14.3 Signals block example	84
15. SignalGroups block	85
15.1 General	85
15.2 SignalGroups block syntax	85
15.3 SignalGroups block example.....	86
15.4 Default attributes value.....	86
15.5 Translation of based data into WaveformChar characters	87
16. PatternExec block.....	88
16.1 General	88
16.2 PatternExec block syntax.....	88
16.3 PatternExec block example.....	89
17. PatternBurst block	89
17.1 General	89
17.2 PatternBurst block syntax	89
17.3 PatternBurst block example	91
18. Timing block and WaveformTable block.....	91
18.1 General	91
18.2 Timing and WaveformTable syntax	92

18.3 Waveform event definitions	95
18.4 Timing and WaveformTable example	98
18.5 Rules for timed event ordering and waveform creation.....	99
18.6 Rules for waveform inheritance.....	101
19. Spec and Selector blocks	102
19.1 General	102
19.2 Spec and selector block syntax	103
19.3 Spec and selector block example	104
20. ScanStructures block	105
20.1 General	105
20.2 ScanStructures block syntax	106
20.3 ScanStructures block example	107
21. STIL Pattern data.....	108
21.1 General	108
21.2 Cyclized data	108
21.3 Multi-bit cyclized data.....	109
21.4 Non-cyclized data	110
21.5 Scan data.....	110
21.6 Pattern labels.....	111
22. STIL Pattern statements.....	111
22.1 Vector (V) statement.....	111
22.2 WaveformTable (W) statement	112
22.3 Condition (C) statement.....	112
22.4 Call statement	113
22.5 Macro statement	113
22.6 Loop statement	113
22.7 MatchLoop statement	114
22.8 Goto statement.....	115
22.9 BreakPoint statements	115
22.10 IDDQTestPoint statement.....	115
22.11 Stop statement.....	116
22.12 ScanChain statement.....	116
22.13 FreeRunningClockStart statement	116
22.14 FreeRunningClockStop statement	117
23. Pattern block.....	117
23.1 General	117
23.2 Pattern block syntax.....	117
23.3 Pattern initialization.....	117
23.4 Pattern examples.....	118
24. Procedures and MacroDefs blocks	118
24.1 General	118
24.2 Procedures block.....	119
24.3 Procedures example.....	120
24.4 MacroDefs block	120
24.5 Scan testing.....	120
24.6 Procedure and Macro data substitution.....	121
24.7 ShiftMultiple.....	125
24.8 ShiftMultiple example	125
Annex A (informative) Glossary	127

Annex B (informative) STIL data model.....	128
Annex C (informative) Binary STIL data format	133
C.1 Binary STIL considerations	133
C.2 Binary STIL conclusions	134
Annex D (informative) LS245 design description.....	137
Annex E (informative) STIL FAQs and language design decisions.....	139
Annex F (informative) 2.0 revision updates	143
F.1 New capabilities	143
F.2 Clarifications	143
Annex G (informative) Bibliography	145

List of Figures

Figure 1—A conduit for transporting data from CAE to ATE	16
Figure 2—STIL usage model	16
Figure 3—Waveforms associated with signal DIR	25
Figure 4—Waveforms associated with the signal OE_	26
Figure 5—Waveform characteristics of WaveformTable pulsed_oe	34
Figure 6—IEEE Std 1149.1-1990 example timings	43
Figure 7—Burst or pipelined data bus.....	47
Figure 8—Serial data.....	48
Figure 9—Design configuration.....	51
Figure 10—Cyclized input data from the events	60
Figure 11—Output values of the bidirectional buses	60
Figure 12—Creating waveforms from events	100
Figure 13—Waveforms extending into the next cycle	101
Figure 14—Using the PatternExec to resolve Spec and Selector information	105
Figure 15—ScanStructures block example	107
Figure B.1—Top level of the STIL data model.....	128
Figure B.2—PatternBurst, PatternExec, and Selector relationships.....	129
Figure B.3—Signal, SignalGroup, and ScanStructures group relationships	130
Figure B.4—Pattern statement relationships	131
Figure B.5—Timing statement relationships.....	132
Figure D.1—LS245 structural model	137
Figure D.2—LS245 Verilog representation	138

List of Tables

Table 1—Hello tester	23
Table 2—STIL statement structure	26
Table 3—Basic LS245	28
Table 4—Mapping of a hex value to the group ABUS_O.....	31
Table 5—Spec timing tests LS245	31
Table 6—LS245 with hypothetical scan.....	35
Table 7—Advanced scan.....	39
Table 8—IEEE Std 1149.1-1990 scan.....	43
Table 9—STIL code to support pipelined data.....	48
Table 10—STIL code to support serial data.....	49
Table 11—DMA_1.stil.....	51
Table 12—CNTR_1.stil	51
Table 13—Q1044.stil	52
Table 14—Stimulus used to generate STIL data	54
Table 15—Stimulus and response data for the LS245.....	54
Table 16—LS245 event data	55
Table 17—LS245 mixed event and cyclized data	58
Table 18—LS245 fully cyclized data	60
Table 19—STIL reserved words	63
Table 20—STIL reserved characters	64
Table 21—SI units.....	69
Table 22—Engineering prefixes.....	69
Table 23—Operators and functions allowed in a timing expression	70
Table 24—STIL name spaces.....	72
Table 25—STIL statement structure	74
Table 26—STIL top-level statements and ordering requirements.....	75
Table 27—Optional top-level statements	76
Table 28—Drive events.....	96
Table 29—Compare events	97
Table 30—Expect events.....	98
Table 31—Unresolved events	98
Table 32—Comparison of Procedures and MacroDefs.....	118
Table 33—Example scan data normalization	124
Table C.1—Comparison of formats	136

IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data

1. Overview

Standard test interface language (STIL) is a standard language that provides an interface between digital test generation tools and test equipment. STIL may be directly generated as an output language of a test generation tool, or it may be used as an intermediate format for subsequent processing. Figure 1 shows STIL usage in a “pipe” format. This is meant solely as a visual analogy to emphasize the high-volume/high-throughput requirements. It is not meant to represent physical structures or implementation requirements.

STIL is a representation of information needed to define digital test operations in manufacturing tests. STIL is not intended to define how the tester implements that information. While the purpose of STIL is to pass test data into the test environment, the overall STIL language is inherently more flexible than any particular tester. Constructs may be used in a STIL file that exceed the capability of a particular tester. In some circumstances, a translator for a particular type of test equipment may be capable of restructuring the data to support that capability on the tester; in other circumstances, separate tools may operate on that data to provide that restructuring. In all circumstances, it is desirable to provide the capability to check the data against the constraints of a tester. This capability is referred to as Tester Rules Checking and is the domain of tools that operate on STIL data. As such, Tester Rules Checking operations are outside the scope of this standard.

Figure 2 shows how STIL fits into the data flow between computer-aided engineering (CAE)/simulation and the test environment. In this figure, STIL is shown as both the input and output of “STIL Manipulation Tools.” STIL represents patterns as a series of cyclized waveforms that are executed sequentially. The waveform representation can be as simple as a “print-on-change” set of events, or a complex set of parameterized events. Hence, tools may be required to manipulate the data according to the requirements of a particular class of device, simulation, or tester. The output of that manipulation is still represented in STIL.

Another issue presented in Figure 2 is the need for data from the tester to be transmitted back to the CAE/simulation environment for the purpose of correlating simulation data to tester data. Although this is recognized as an important aspect of testing digital devices, it does not represent the data volume that the patterns themselves do, and is not specifically supported in this version of the standard.

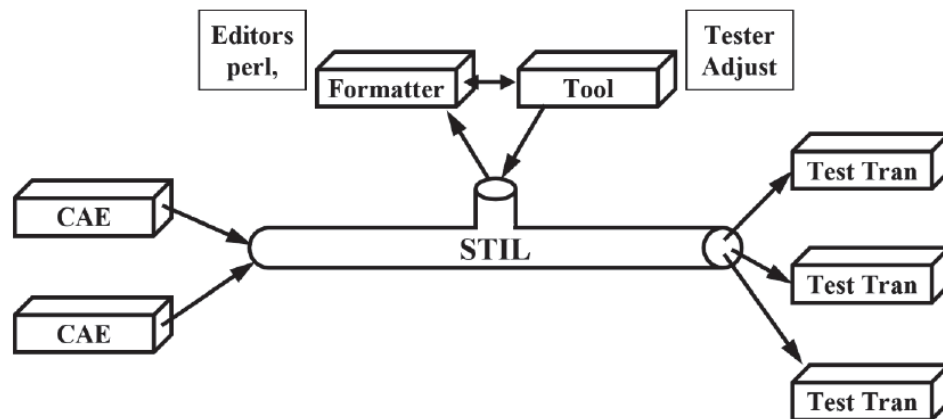


Figure 1—A conduit for transporting data from CAE to ATE

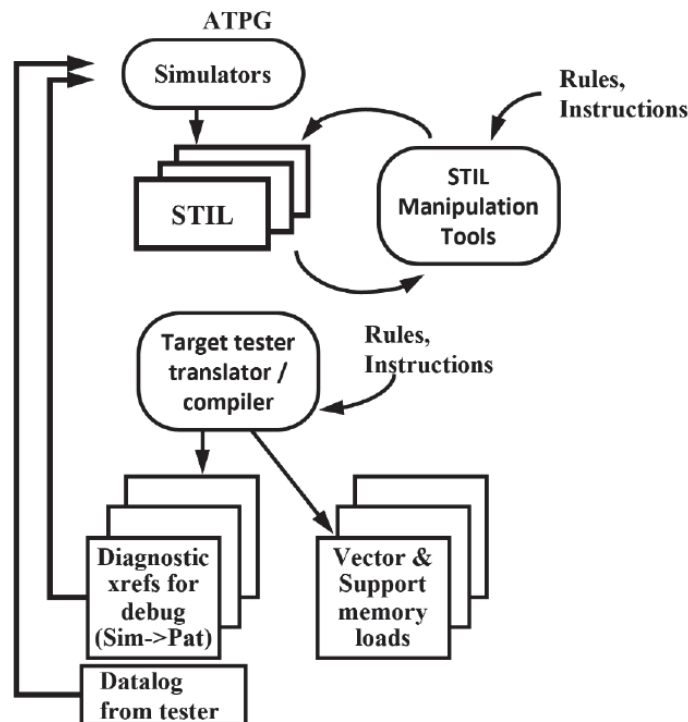


Figure 2—STIL usage model

1.1 Scope

This standard defines a test description language that [achieves the following goals](#):

- a) Facilitates the transfer of large volumes of digital test vector data from [computer-aided engineering \(CAE\)](#) environments to automated test equipment (ATE) environments. 
- b) Specifies pattern, format, and timing information sufficient to define the application of digital test vectors to a device under test (DUT). 
- c) Supports the volume of test vector data generated from structured tests such as scan/automatic test pattern generation (ATPG), integral test techniques such as built-in self test (BIST), and functional test specifications for IC designs and their assemblies, in a format optimized for application in ATE environments.

In setting the scope for any standard, some issues are defined to not be pertinent to the initial project. The following is a partial list of issues that were dropped from the scope of this initial project:

- Levels: A key aspect of a digital test program is the ability to establish voltage and current parameters (levels) for signals under test. Level handling is not explicitly defined in the current standard, as this information is both compact (not presenting a transportation issue) and commonly established independently of digital test data, requiring different support mechanisms outside the current scope of this standard. Termination values may affect levels.
- Diagnostic/fault-tracing information: The goal of this standard is to optimally present data that needs to be moved onto ATE. While diagnostic data, fault identification data, and macro/design element correspondence data can fall into this category (and is often fairly large), this standard is also focused on integrated circuit and assemblies test, and most debug/failure analysis occurs separately from the ATE for these structures. Note that return of failure information (for off-ATE analysis) is also not part of the standard as currently defined.
- Datalogging mechanisms, formatting, and control usually are not defined as part of this current standard.

- Parametric tests are not defined as an integral part of this standard, except for optional pattern labels that identify potential locations for parametric tests, such as IDDQ tests or alternating current (ac) timing tests.
- Program flow: Test sequencing and ordering are not defined as part of the current standard except as necessary to define collections of digital patterns meant to execute as a unit.
- Binning constructs are not part of the current standard.
- Analog or mixed-signal test: While this is an area of concern for many participants, at this point transfer of analog test data does not contribute to the same transportation issue seen with digital data.
- Algorithmic pattern constructs (such as sequences commonly used for memory test) are not currently defined as part of the standard.
- Parallel test/multisite test constructs are not an integral part of the current environment.
- User input and user control/options are not part of the current standard.
- Characterization tools, such as shmoo plots, are not defined as part of the current standard.

1.2 Purpose

This standard addresses a need in the integrated circuit (IC)[†] test industry to define a standard mechanism for transferring the large volumes of digital test data from the generation environment through to test. The environment today contains unique output formats of existing CAE tools, individual test environments of IC manufacturers, and proprietary IC ATE input interfaces. As each of these three arenas solves individual problems, together they have created a morass of interfaces, translators, and software environments that provide no opportunity to leverage common goals and result in much wasted efforts re-engineering solutions. As device density increases, the magnitude of test data threatens to shift the test bottleneck from the generation process to the processes necessary solely to maintain and transport this data. These two factors threaten to eliminate any productive work performed in this area unless a viable standard is defined.

[†]The use of this term in this standard is meant only as a point of reference and not to indicate an explicit limitation or restriction of focus.

With a common standard for CAE and IC ATE environments, the generation, movement, and processing of this test data is greatly facilitated. This standard also allows for immediate access to test equipment supporting this standard, which benefits both ATE and IC vendors reviewing this equipment.

This standard also serves as a catalyst for the development of a set of standard third party interface tools to both test and design aspects of IC device generation.

1.3 Word usage

The word *shall* indicates mandatory requirements strictly to be followed in order to conform to the standard and from which no deviation is permitted (*shall* equals *is required to*).^{6,7}

The word *should* indicates that among several possibilities one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required (*should* equals *is recommended that*).

The word *may* is used to indicate a course of action permissible within the limits of the standard (*may* equals *is permitted to*).

The word *can* is used for statements of possibility and capability, whether material, physical, or causal (*can* equals *is able to*).

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

⁶ The use of the word *must* is deprecated and cannot be used when stating mandatory requirements; *must* is used only to describe unavoidable situations.

⁷ The use of *will* is deprecated and cannot be used when stating mandatory requirements; *will* is only used in statements of fact.

~~This standard shall be used in conjunction with the following standards. If the following publications are superseded by an approved revision, the revision shall apply~~

~~IEEE Std 100-1996, The IEEE Standard Dictionary of Electrical and Electronics Terms, Sixth Edition.²~~

~~IEEE Std 260.1TM-2004 1993, American National Standard Letter Symbols for Units of Measurement (SI Units, Customary Inch-Pound Units, and Certain Other Units).^{8,9}~~

~~ISO 2955:1983, Information processing—Representation of SI and other units in systems with limited character sets.^{10 3}~~

~~ISO/IEC 9899:2018 1990, Programming languages—C.⁴~~

^{8,2} IEEE publications are available from the Institute of Electrical and Electronics Engineers, [Inc., 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA](http://standards.ieee.org/) (<http://standards.ieee.org/>).

⁹ [The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.](#)

¹⁰ [ISO publications are available from the International Organization for Standardization \(<https://www.iso.org/>\) and the American National Standards Institute \(<https://www.ansi.org/>\).](#)

¹¹ [IEEE Standards Dictionary Online is available at: <http://dictionary.ieee.org>. An IEEE account is required for access to the dictionary, and one can be created at no charge on the dictionary sign-in page.](#)

³~~ISO publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembe, CH 1211, Genève 20, Switzerland/Suisse (<http://www.iso.ch/>). ISO publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA (<http://www.ansi.org/>).~~

⁴~~IEC publications are available from the Sales Department of the International Electrotechnical Commission, Case Postale 131, 3, rue de Varembe, CH 1211, Genève 20, Switzerland/Suisse (<http://www.iec.ch/>). IEC publications are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.~~

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this ~~document~~ ~~standard~~, the following terms and definitions apply. ~~Additional terminology specific to this standard is found in Annex A. IEEE Std 100-1996, The IEEE Standards Standard Dictionary Online of Electrical and Electronics Terms, Sixth Edition,~~ should be ~~consulted~~ ~~referenced~~ for terms not defined in this ~~clause.~~¹¹ ~~document.~~

automatic test pattern generator (ATPG): Any tool that generates test information for a device based on structural analysis of the device.

breakpoint: A position within a pattern set where the pattern may be segmented into multiple independent bursts while still achieving predictable behavior of the device.

built-in self-test (BIST): A ~~design practice in which~~ test ~~logic is incorporated into the~~ ~~paradigm that incorporates~~ circuitry ~~of a semiconductor in the in the~~ device ~~for executing and resolving test information about the device.~~

burst: Tester execution of a pattern or set of patterns. Generally controlled by “start” and “stop” definitions.

computer-aided engineering (CAE): A computer-based set of tools to assist in the design and development of integrated circuits.

cyclize: To drive a tester, data ~~shall~~ ~~must~~ be provided in uniform, consistent, ~~and~~ repeatable collections. These collections are termed “cycles,” or “tester cycles.” The process of constructing these collections, generally from simulation environments, is called “cyclizing.”

device: A reference to an integrated circuit or other design structure.

device under test (DUT): The device to be placed in a test fixture and tested.

float-state: A logic value that indicates the lack of an active drive condition, generally used in an environment with multiple drivers connected to a single signal, and commonly referenced in digital simulation as a “Z” state.

functional vector: A pattern generated to exercise a device's functional behavior. Generally defined to run the device at system speeds to verify system behavior of a design. *Contrast with:* **structural vectors**.

IDDQ: Current measurement taken at the ground rail during quiescent operation.

incremental vector: A representation of test vectors containing only the changing signals and new signal values in each vector. Parallel vectors can be generated from incremental vectors by maintaining test-specified state information for signals that did not change.

metatype: A collection of defined linguistic entities that share some common features.

NOTE— ÷ In this standard, all defined metatypes represent collections of entities, which may be used interchangeably in the language.¹²

newline: The character or characters necessary to generate the start of the next line of **ASCII** text. May also be known as a carriage-return (CR), linefeed (LF), or a CR-LF combination.

parallel vector: A representation specifying a set of waveforms across all primary signals, to be applied to those signals in a parallel fashion (i.e., simultaneously).

parametric test: A test that is performed to verify device behavior such as output drive current, input leakage current, or output voltage.

pattern: One or more vectors comprising a functionality test for a specific portion of a device under test (DUT).

primary signal: A signal at the interface between the physical device and the physical tester. Any and all information meant for test is defined on these signals. ÷ Test translators need to process these signals only.

pseudo signal: A signal other than that at the interface between the device and the tester. This includes internal signals, derived signals, and any other signals that may be required by tools other than test translators to generate tests or test constructs.

¹² Notes in text, tables, and figures are given for information only and do not contain requirements needed to implement the standard.

scan input signal: A primary signal which may be used to serially precondition the scan register latches of the [device under test \(DUT\)](#).

scan output signal: A primary signal which may be used to serially observe the contents of the scan register latches of the [device under test \(DUT\)](#).

scan test methodology: A test methodology that utilizes shift register latches to precondition and observe modeled faults within the [device under test \(DUT\)](#). Scan tests typically consist of a serial preconditioning (load via scan inputs), parallel vectors to clock/transition the DUT, and then a serial observation (unload via the scan outputs).

scan vectors: A representation of test information containing lists of states that are to be shifted into or out of the scan pins on the device.

NOTE — ÷ Scan vectors imply the use of scan test methodology in the design of the device under test.

signal: A point in the design from which a stimulus may be directly applied or a response directly measured.

standard test interface language (STIL): A syntax for the description of device stimulus and expected response used for stimulus development, as well as input to automated test equipment (ATE).

structural vectors: A pattern generated to exercise a device's structural elements [e.g., scan-based [automatic test pattern generator](#) (ATPG) test generation]. *Contrast with:* **functional vectors**.

T0 (pronounced "tee-zero"): A reference to a [system](#) **MASTER** clock that synchronizes all events across all signals to a common starting point. Initiates the start of each test vector.

termination: A constant impedance and digital logic state that a signal is held at during some or all of a test.

tester cycle: *See:* **vector**.

valid compare: A condition on output response when the precise state of the response is not important to the test, but the fact that the output is a valid state value is pertinent.

valid input: A condition on input stimulus when the state of that stimulus does ~~will~~ not affect the current test. In the simulator perspective, this condition is often identified as an unknown, or X, state.

vector: Every signal's stimuli/response to be applied/observed in the smallest integral "step" of a device test. Contains a collection of waveforms to be applied to the primary signals. *See: T0.*

waveform: A stream of defined events containing both state and timing information.

waveshape: A stream of defined states or transitions with no associated timing.

3.2 Acronyms and abbreviations

ATE	automated test equipment
ATPG	automatic test pattern generator
BIST	built-in self-test
BNF	Backus-Naur form
CAE	computer-aided engineering
DFT	design for test
DAT	direct access test
DMA	direct memory access
DUT	device under test
FSM	finite state machine
IC	integrated circuit
I/O	input/output
LSB	least significant bit
MSB	most significant bit
TAP	test access port
TCK	test clock

TDI	test data in
TDO	test data out
TMS	test mode select

4. Structure of this standard

This standard is partitioned into several clauses to assist those who are just discovering the language to learn the constructs and capabilities of [standard test interface language \(STIL\)](#), and to facilitate those experienced with the language to find the particular element they need.

Clause 5 is structured as an informative tutorial to the language, and serves to introduce STIL concepts, starting with the basics and expanding into special purpose or more elaborate constructs. This clause elaborates on what is happening (and why); however, it is not intended to be a complete presentation on each construct, nor is it a normative part of the specification.

Following the tutorial are the language definition clauses. These clauses present the entire language, with all requirements and capabilities delineated completely.

The following conventions are used in this standard. Different fonts are used as follows:

- a) SMALL CAP TEXT is used to indicate user data;
- b) courier text is used to indicate code examples.

In the syntax definitions, [different formatting indicators are used as follows](#):

- a) SMALL CAP TEXT is used to indicate user data;
- b) **bold text** is used to indicate keywords;
- c) *italic text* is used to reference metatypes;
- d) () indicates optional syntax which may be used 0 or 1 time;
- e) ()+ indicates syntax which may be used 1 or more times;
- f) ()* indicates optional syntax which may be used 0 or more times;
- g) <> indicates multiple choice arguments or syntax.

~~In the syntax explanations, the verb “shall” is used to indicate 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 files (“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 in the dashed list above (Clause 4).~~

~~Waveforms represented in the diagrams use symbols defined in Table 9 through Table 12. Use the information in these tables to help understand waveform diagrams.~~

5. STIL orientation and capabilities tutorial **(informative)**

5.1 General

This clause presents an overview of STIL through a layered tutorial that explains the language constructs. This clause is informative and is not a part of IEEE Std 1450-1999, ~~Standard Test Interface Language~~, (STIL) for digital test vector data.

5.2 Hello tester

[Table 1](#) ~~Figure 3~~ represents a complete STIL program to exercise a subset of behavior for an octal bus transceiver design, modeled after a TTL LS245. Details of this design are found in Annex [D](#) ~~E~~. This example defines the LS245 as “unidirectional.” To simplify this example, the “A” bus signals are defined as inputs, and the “B” bus signals are defined as outputs. [Table 1](#) ~~Figure 3~~ is annotated and explanations (notes) for each of the marked sections follow the [table figure](#).

NOTE—[Table Figure](#) notes follow all of the annotated [tables figures](#) in this standard.

Table 1 Figure 3—Hello tester

```
a STIL 1.0;

b Signals {
  DIR In;
  OE_ In;
  A0 In; A1 In; A2 In; A3 In;
  A4 In; A5 In; A6 In; A7 In;
  B0 Out; B1 Out; B2 Out; B3 Out;
  B4 Out; B5 Out; B6 Out; B7 Out;
}

c SignalGroups {
  ABUS='A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0';
  BBUS='B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0';
  ALL ='DIR + OE_ + ABUS + BBUS';
}

d Timing "hello tester timing" {
  WaveformTable one {
e    Period '500ns';
    Waveforms {
f      DIR { 01 { '0ns' ForceDown/ForceUp; }}
      OE_ { 01 { '0ns' ForceUp; '200ns' ForceDown/ForceUp;
                  '300ns' ForceUp; }}
      ABUS { 01 { '10ns' ForceDown/ForceUp; }}
      BBUS { HLZ { '0ns' ForceOff; '0ns' CompareUnknown;
                  '260ns' CompareHighWindow/CompareLowWindow/CompareOffWindow;
                  '280ns' CompareUnknown;}}
    }
g  } // end WaveformTable one
  } // end Timing "hello tester timing"

h PatternBurst "hello tester burst" {
  PatList { "hello tester pattern"; {
  } // end PatternBurst "hello tester burst"

i PatternExec {
  Timing "hello tester timing";
  PatternBurst "hello tester burst";
  } //end PatternExec

j Pattern "hello tester pattern" {
  W one;
  V { ALL=0000000000LLLLLLL; }
  V { ALL=0010000000HLLLLLLL; }
  V { ALL=0001000000LHLLLLLL; }
  V { ALL=0000100000LLHLLLLL; }
  V { ALL=0000010000LLLHLLLL; }
  V { ALL=0000001000LLLLHLLL; }
  V { ALL=0000000100LLLLLHLL; }
  V { ALL=0000000010LLLLLLHL; }
  V { ALL=0000000001LLLLLLLH; }
  } //end Pattern "hello tester pattern"
```

[\(New\)](#)

① STIL 1.0;

② Signals {
 DIR In;
 OE_ In;
 A0 In; A1 In; A2 In; A3 In;
 A4 In; A5 In; A6 In; A7 In;
 B0 Out; B1 Out; B2 Out; B3 Out;
 B4 Out; B5 Out; B6 Out; B7 Out;
 }

③ SignalGroups {
 ABUS='A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0';
 BBUS='B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0';
 ALL = 'DIR + OE_ + ABUS + BBUS';
 }

④ Timing "hello tester timing" {
 WaveformTable one {
 ⑤ Period '500ns';
 Waveforms {
 ⑥ DIR { 01 { '0ns' ForceDown/ForceUp; }}
 OE_ { 01 { '0ns' ForceUp; '200ns' ForceDown/ForceUp;
 '300ns' ForceUp; }}
 ABUS { 01 { '10ns' ForceDown/ForceUp; }}
 BBUS { HLZ { '0ns' ForceOff; '0ns' CompareUnknown;
 '260ns' CompareHighWindow/CompareLowWindow/CompareOffWindow;
 '280ns' CompareUnknown; }}
 }
 } // end WaveformTable one ⑦
 } // end Timing "hello tester timing"

⑧ PatternBurst "hello tester burst" {
 PatList { "hello tester pattern";
 }
 } // end PatternBurst "hello tester burst"

⑨ PatternExec {
 Timing "hello tester timing";
 PatternBurst "hello tester burst";
 }
 } //end PatternExec

⑩ Pattern "hello tester pattern" {
 W one;
 V { ALL=0000000000LLLLLLLL; }
 V { ALL=0010000000HLLLLLLL; }
 V { ALL=0001000000LHLLLLLL; }
 V { ALL=0000100000LLHLLLLL; }
 V { ALL=0000010000LLLHLLLL; }
 V { ALL=0000001000LLLLHLLL; }
 V { ALL=0000000100LLLLLHLL; }
 V { ALL=0000000010LLLLLLHL; }
 V { ALL=0000000001LLLLLLHH; }
 }
 } //end Pattern "hello tester pattern"

The numbers in the circles (e.g., ①) correspond to the figure notes that follow.

(Deleted)

Notes for Figure 3.

^a ~~NOTE 1~~—The very first statement in a STIL file is the **STIL** statement. This statement defines the version of the STIL language following this statement.

^b ~~NOTE 2~~—The **Signals** block defines a name for each signal used in the test vectors and identifies the signal type, such as In, Out, InOut, Supply, or Pseudo. Remember that in this example the bidirectional busses of the LS245 design are defined as unidirectional and, therefore, only In and Out are used here.

^c ~~NOTE 3~~—The **SignalGroups** block defines an ordered set of signals to be referenced in subsequent operations. In this example, three groups are defined: a collection of all bits of the “A” bus, called ABUS; a collection of all bits of the “B” bus, called BBUS; and a collection of all signals in the design called ALL. ALL has been defined using the two previous group definitions. The operators “+” and “-” are used to define these ordered groups in objects called “pin expressions.”

^d ~~NOTE 4~~—The **Timing** block defines sets of “WaveformTables.” Each WaveformTable defines the waveforms to be applied to each signal used in a vector. After the Timing keyword is the quoted string “Hello Tester Timing.” This quoted string becomes the name of this Timing block. By enclosing the name with double-quotes, characters such as spaces can be made to be part of the name.

^e ~~NOTE 5~~—The first statement in a WaveformTable is the period of the test vector to be applied to all signals. All signals defined in a single WaveformTable shall ~~must~~ have the same period. In this example, the tester period is 500 ns long.

Each signal may have several different waveforms defined in a single WaveformTable. Each waveform defined for a signal is ~~will be~~ referenced with a single character, called a WaveformChar, or “WFC.” Within each WaveformTable, each signal’s WaveformChars shall ~~must~~ be unique across all waveforms defined for the signal. However, different signals may define the same WaveformChar for different waveforms.

A waveform needs some explanation. In STIL, a waveform is a series of “time” and “event” pairs. Each pair is defined with a single STIL statement. These statements are also referred to as “timed events.” The “event” may be a special single character defined to have a particular operation, or it may be a longer identifier as used in this example. This example used the events “ForceDown,” “ForceUp,” “ForceOff,” “CompareUnknown,” “CompareHighWindow,” “CompareLowWindow,” and “CompareOffWindow.” “ForceDown” and “ForceUp” are input or drive events; “ForceDown” forces a logic low on an input, and “ForceUp” forces a logic high. “ForceOff” forces a logic float-state, or turns off any input drivers. “CompareHighWindow,” “CompareLowWindow,” “CompareOffWindow,” and “CompareUnknown” are output, or expect, events. “CompareHighWindow” expects a logic high, “CompareLowWindow” expects a logic low, and “CompareOffWindow” expects a logic float-state value. To close a window strobe, the event “CompareUnknown” is used.

There are four distinct types of signals in this design. Each has its own waveform to represent the input or output information required to test this design.

NOTE 6—The first waveform definition is for the signal DIR. This signal controls the “direction” of the bused signals, which is fixed in this test. Even though it is fixed, information is defined for signal DIR to allow this signal to be driven high or driven low at the start of each test cycle. Figure 3 4 shows graphically the two waveforms defined for this signal and the STIL syntax.

Because both waveforms have the same timing, they can be merged into a single STIL statement. This shorthand syntax allows multiple WaveformChars to be defined to the same event in the waveform, with states for each WaveformChar to apply at that time. The relationship of WaveformChar to event characters is direct: the first WaveformChar (in the example above, “0”) maps to the first waveform event (“ForceDown”), the second (“1”) maps to the second event (“ForceUp”), and so on for each WaveformChar present. Note that a slash shall ~~must~~ be present to separate the event references when more than one is present.

The signal OE_ has additional events defined to create a pulsed behavior. The merging process for this signal is not as intuitive as DIR, as the process here requires defining events that do not cause a change of state on the signal. The only reason to define these events is to support a single waveform definition for this signal. [Figure 4 shows graphically the two waveforms defined for this signal and the STIL syntax.](#)

The ABUS signals are defined similarly to DIR, except that they are offset 10 ns into the vector boundary. Note the use of a Group here to reference a collection of signals to be defined.

The BBUS signals are defined as outputs; at entry to the cycle, any test drivers are explicitly turned off with the "ForceOff" event. The WaveformChar characters H, L, and Z are mapped to the expected states "CompareHighWindow," "CompareLowWindow," and "CompareOffWindow," respectively. The strobe window is opened at 260 ns. At 280 ns, the strobe is closed with a "CompareUnknown" event.

NOTE 7—STIL supports two styles of comments: "block comments," which are delimited by a "/*" and "*/", and "comments to the newline," which are delimited by a "/*" and terminated with a newline. The comments annotating the closing braces in this example use the style of comments to the newline.

NOTE 8—The **PatternBurst** block defines a collection of pattern names to be executed sequentially. (In this example, there is only one pattern defined.) All patterns defined in a single PatternBurst are executed under a similar context, the context being defined by the subsequent PatternExec statement. The references to pattern names in this block are one of the few forward references allowed in STIL; patterns are not defined until the end of the STIL data.

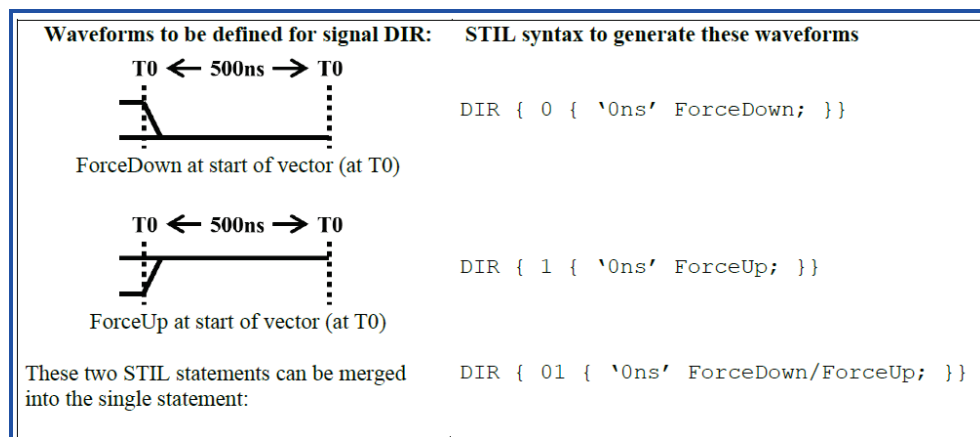
NOTE 9—The **PatternExec** block defines how PatternBurst and Timing information is assembled to create the set of tests to execute. The references in this block to Timing and PatternBurst names **must** have been defined before this block.

NOTE 10—Finally, the pattern data is defined. Pattern data constitutes the bulk of data in the STIL data set, and is generally processed one- vector- at- a time. In order to support processing this data as it is read, it is necessary to define pattern data as the last data in a STIL test environment.

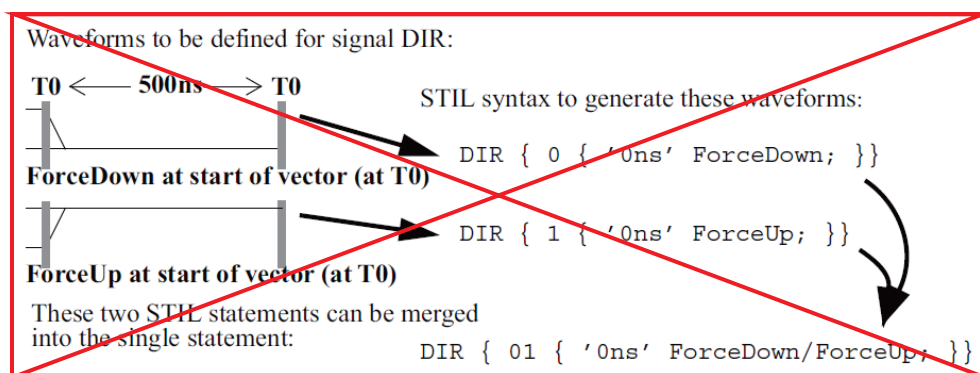
In this example, the first statement in the **Pattern** block is a reference to a WaveformTable. The following vectors (until another 'W' statement) uses ~~will use~~ the timing defined in the WaveformTable named "one."

Note that while this pattern contains references to WaveformChars, and to names of WaveformTables, it does not contain any direct references to the Timing block. This resolution is provided by the PatternExec statement. The PatternExec can define references to different Timing blocks; as long as the WaveformChars and WaveformTable names are defined in the referenced timing set, they can be applied to these same patterns.

Each V statement defines one test vector. In this example, each Vector defines the state to be applied to each signal using the group reference ALL. The declaration order of signals in the group ALL is critical, as the mapping of WaveformChars to signals in the group is performed linearly.

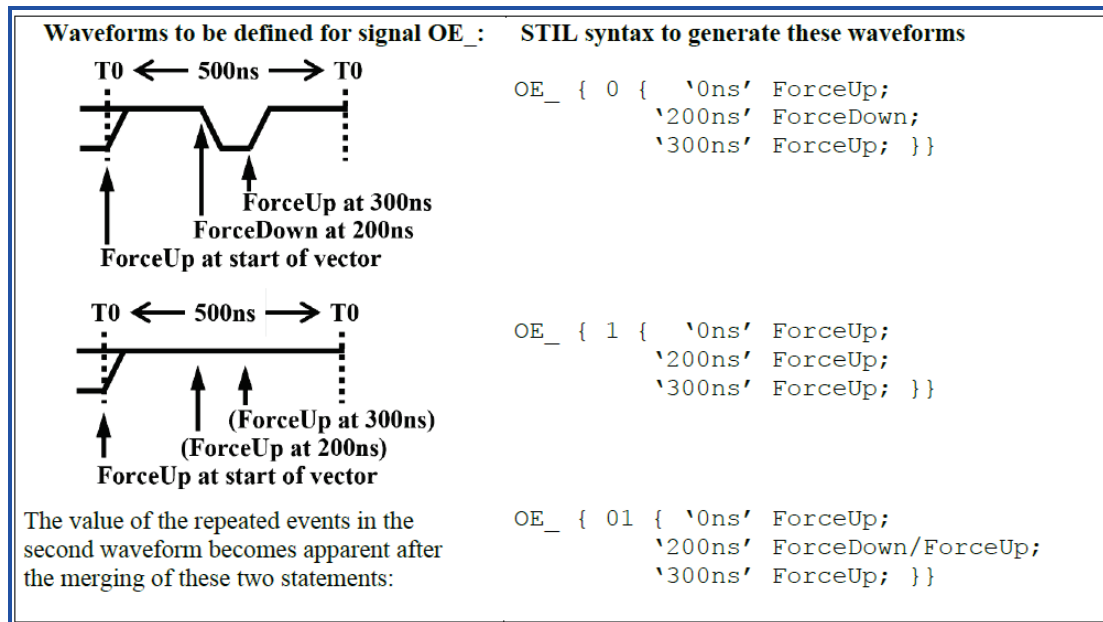


(New)

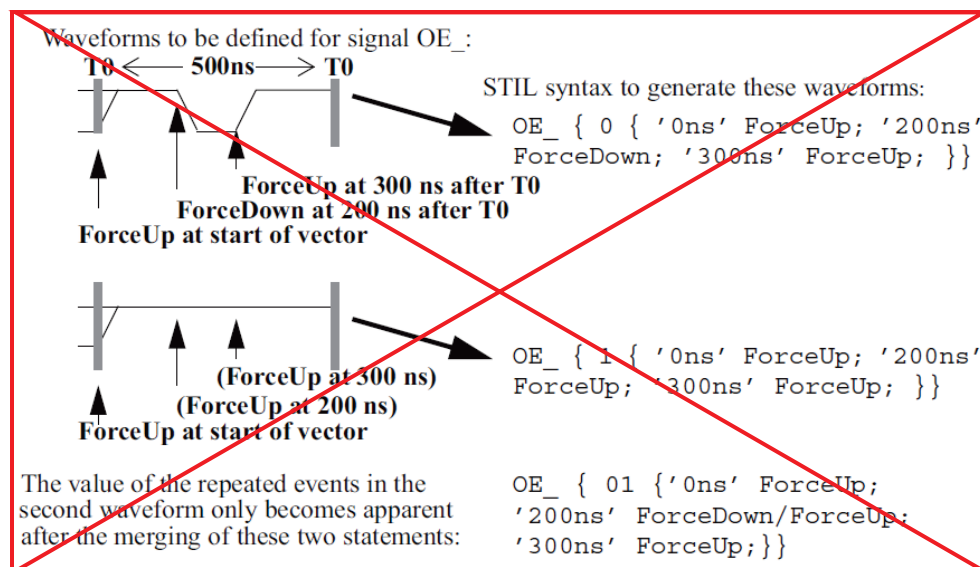


(Deleted)

Figure 3 4—Waveforms associated with signal DIR



(New)



(Deleted)

Figure 4 5—Waveforms associated with the signal OE_

5.2.1 STIL grammatical constructs

STIL has two basic grammatical constructs for statements in the language. The first is a “simple statement,” featuring a keyword, zero or more other tokens, and terminated by a semicolon. The second is a “block statement,” which ~~again~~ starts with a keyword, may again be followed by zero or more tokens, and then has an open-brace. After the open-brace, additional STIL language statements may occur. This statement is terminated by a closing brace. These two statement formats are shown in [Table 2 Figure-6](#).

[Table 2 Figure-6](#)—STIL statement structure

Simple statement: Keyword (OPTIONAL_TOKENS)*;
Block statement: Keyword (OPTIONAL_TOKENS)* { (OPTIONAL_MORE_STATEMENTS)* }

[Table 2 Figure-6](#) is intended to represent a simplification of the STIL syntax. Some statements, such as the assignment statements in [Table 1 Figure-3](#) (ALL=0000000000LLLLLLLL), also require the “=” sign to be present.

STIL is a case-sensitive language. All STIL keywords start with an uppercase letter, and some may have additional uppercase letters inside.

5.2.2 Complexity and language subsets

The previous example of STIL data has been reduced to present the basic language constructs only. As this tutorial progresses through additional examples, different aspects of the STIL language [are](#) ~~will be~~ presented.

There are many ways in which digital test information may be developed or provided. Some types of tests may be more concerned about testing device specifications. Such tests may not care about how the device implements those specifications, only that the specifications are satisfied. Other types of tests may be concerned with device operation, such as functional tests, and still others may only be testing the device from a structural perspective (i.e., the elements present in the device and their interconnections) and not even have a concept of how the device is meant to be used. STIL supports all of these perspectives.

Another issue with passing information into the test environment is the definition of the test environment itself. Test equipment is varied in performance, capability, and capacity. If the goal of STIL is to provide information to be used in the test environment, how does the language [help](#) ensure that this can happen? This is a major issue in the test industry, and a standard language is not going to address the problem. It was deemed critical that STIL attempt to represent tester capability relevant to digital IC and assembly test, in order to provide a mechanism to move information onto “capable” test environments, and not to constrain the language to the lowest common denominator of test capability.

Intentionally, STIL does not cause or enforce constraints on what can be represented. For example, you could legally specify a waveform with eight events. Tools reading STIL, such as a tester vendor's pattern compiler, ~~will~~ enforce the target tester constraints, or possibly translate the request into something that the tester can support (e.g., two four-event waveforms multiplexed together).

These issues, and several others that [are](#) ~~will be~~ presented as STIL is discussed in this standard, lead to the inevitable conclusion that STIL, in some aspects, is rather complex. The important perspective that should be maintained is that not all of the complexity of STIL may be needed to represent device test information. Use only those constructs that are appropriate to the needs.

While this tutorial presents STIL in a “phased” aspect, from “simple” or mandatory information to more “advanced” constructs, it is important to remember that there are no actual classifications in the language. This presentation is structured in this fashion solely to facilitate presentation of the concepts.

5.3 Basic LS245

The previous example demonstrated a subset of the LS245 behavior. In [Table 3](#), ~~this example, we present~~ a complete STIL test for this device [is presented](#).

Table 3 ~~Figure 7~~—Basic LS245

```
STIL 1.0;

Signals {
  DIR In;
  OE_ In;
a  A0 InOut; A1 InOut; A2 InOut; A3 InOut;
  A4 InOut; A5 InOut; A6 InOut; A7 InOut;
  B0 InOut; B1 InOut; B2 InOut; B3 InOut;
  B4 InOut; B5 InOut; B6 InOut; B7 InOut;
}

SignalGroups {
  ABUS = 'A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0';
  BBUS = 'B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0';
  ALL = 'DIR + OE_ + ABUS + BBUS';
}

b SignalGroups more {
c  ABUS_I = 'ABUS' { Base Hex 01; }
  BBUS_I = 'BBUS' { Base Hex 01; }
  ABUS_O = 'ABUS' { Base Hex LHZX; }
  BBUS_O = 'BBUS' { Base Hex LHZX; }
}

Timing basic {
  WaveformTable one {
    Period '500ns';
    Waveforms {
      DIR { 01 { '0ns' ForceDown/ForceUp; }}
      OE_ { 01 { '0ns' ForceUp; '200ns' ForceDown/ForceUp;
                '300ns' ForceUp; }}
      ABUS { 01 { '10ns' ForceDown/ForceUp; }
            LHZX{ '0ns' ForceOff;'0ns' CompareUnknown; '260ns'
d CompareLowWindow/CompareHighWindow/CompareOffWindow/CompareUnknown;
                '280ns' CompareUnknown;}}
      BBUS { 01 { '10ns' ForceDown/ForceUp; }
            LHZX{ '0ns' ForceOff;'0ns' CompareUnknown; '260ns'
CompareLowWindow/CompareHighWindow/CompareOffWindow/CompareUnknown;
                '280ns' CompareUnknown;}}
    } // end Waveforms
  } // end WaveformTable one
} // end Timing basic
e PatternBurst basic_burst {
  SignalGroups more;
  PatList { basic; }
} //end PatternBurst basic

PatternExec {
  Timing basic;
  PatternBurst basic_burst;
} //end PatternExec

Pattern basic {
  W one;
// No default states defined;
```

[\(New\)](#)

	// the first vector specifies states on all signals.
f	V { ALL=00ZZZZZZZZXXXXXXX; }
g	V { ABUS_I=00;BBUS_O=0000; } V { ABUS_I=80;BBUS_O=4000; } V { ABUS_I=40;BBUS_O=1000; } V { ABUS_I=20;BBUS_O=0400; } V { ABUS_I=10;BBUS_O=0100; } V { ABUS_I=08;BBUS_O=0040; } V { ABUS_I=04;BBUS_O=0010; } V { ABUS_I=02;BBUS_O=0004; } V { ABUS_I=01;BBUS_O=0001; }
h	V { OE_=1; BBUS_O=FFFF; } V { DIR=1;OE_=0;ABUS_O=FFFF;BBUS_O=AAAA; } V { ABUS_O=0000;BBUS_I=00; } V { ABUS_O=0001;BBUS_I=01; } V { ABUS_O=0004;BBUS_I=02; } V { ABUS_O=0010;BBUS_I=04; } V { ABUS_O=0040;BBUS_I=08; } V { ABUS_O=0100;BBUS_I=10; } V { ABUS_O=0400;BBUS_I=20; } V { ABUS_O=1000;BBUS_I=40; } V { ABUS_O=4000;BBUS_I=80; } V { OE_=1; ABUS_O=FFFF; } } //end Pattern basic

[\(New\)](#)

STIL 1.0;

The numbers in the circles (e.g., ①) correspond to the figure notes that follow.

```
Signals {  
  DIR In;  
  OE_ In;  
  A0 InOut; A1 InOut; A2 InOut; A3 InOut; ①  
  A4 InOut; A5 InOut; A6 InOut; A7 InOut;  
  B0 InOut; B1 InOut; B2 InOut; B3 InOut;  
  B4 InOut; B5 InOut; B6 InOut; B7 InOut;  
}
```

```
SignalGroups {  
  ABUS = 'A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0';  
  BBUS = 'B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0';  
  ALL = 'DIR + OE_ + ABUS + BBUS';  
}
```

```
SignalGroups more {  
  ABUS_I = 'ABUS' { Base Hex 01; } ②  
  BBUS_I = 'BBUS' { Base Hex 01; }  
  ABUS_O = 'ABUS' { Base Hex LHZX; }  
  BBUS_O = 'BBUS' { Base Hex LHZX; }  
}
```

```
Timing basic {  
  WaveformTable one {  
    Period '500ns';  
    Waveforms {  
      DIR { 01 { '0ns' ForceDown/ForceUp; }}  
      OE_ { 01 { '0ns' ForceUp; '200ns' ForceDown/ForceUp;  
                '300ns' ForceUp; }}  
      ④ ABUS { 01 { '10ns' ForceDown/ForceUp; }  
              LHZX { '0ns' ForceOff; '0ns' CompareUnknown; '260ns'  
CompareLowWindow/CompareHighWindow/CompareOffWindow/CompareUnknown;  
                '280ns' CompareUnknown; }}  
      BBUS { 01 { '10ns' ForceDown/ForceUp; }  
              LHZX { '0ns' ForceOff; '0ns' CompareUnknown; '260ns'  
CompareLowWindow/CompareHighWindow/CompareOffWindow/CompareUnknown;  
                '280ns' CompareUnknown; }}  
    } // end Waveforms  
  } // end WaveformTable one  
} // end Timing basic
```

(Deleted)

```

PatternBurst basic_burst {
    SignalGroups more;
    PatList { basic; }
} //end PatternBurst basic
PatternExec {
    Timing basic;
    PatternBurst basic_burst;
} //end PatternExec

Pattern basic {
    W one;
    // No default states defined;
    // the first vector must specify states on all signals.
    6 V { ALL=00ZZZZZZZZXXXXXXXX; }
    7 V { ABUS_I=00;BBUS_O=0000; }
    V { ABUS_I=80;BBUS_O=4000; }
    V { ABUS_I=40;BBUS_O=1000; }
    V { ABUS_I=20;BBUS_O=0400; }
    V { ABUS_I=10;BBUS_O=0100; }
    V { ABUS_I=08;BBUS_O=0040; }
    V { ABUS_I=04;BBUS_O=0010; }
    V { ABUS_I=02;BBUS_O=0004; }
    V { ABUS_I=01;BBUS_O=0001; }

    8 V { OE_=1; BBUS_O=FFFF; }
    V { DIR=1;OE_=0;ABUS_O=FFFF;BBUS_O=AAAA; }

    V { ABUS_O=0000;BBUS_I=00; }
    V { ABUS_O=0001;BBUS_I=01; }
    V { ABUS_O=0004;BBUS_I=02; }
    V { ABUS_O=0010;BBUS_I=04; }
    V { ABUS_O=0040;BBUS_I=08; }
    V { ABUS_O=0100;BBUS_I=10; }
    V { ABUS_O=0400;BBUS_I=20; }
    V { ABUS_O=1000;BBUS_I=40; }
    V { ABUS_O=4000;BBUS_I=80; }

    V { OE_=1; ABUS_O=FFFF; }
} //end Pattern basic

```

(Deleted)

Figure 7—Basic LS245 (continued)

Notes for Figure 7:

^a ~~NOTE 1~~—In this example, the “A” and “B” buses are now defined as bidirectional, or InOut in STIL terminology.

^b ~~NOTE 2~~—Another SignalGroup definition has been added to this example. This SignalGroup has a domain name (“more” without the quotes) after the SignalGroup keyword. In STIL, names may optionally occur before the opening brace of a block section. Named blocks are referenced differently than unnamed blocks. Unnamed blocks are considered to contain “global” information; the information defined in that block may be used by any other sections after that block. Named blocks are “local” information; in order to use that information, that domain name shall ~~must~~ be explicitly referenced in a block after that declaration. The referencing mechanism for Timing blocks was already presented; the referencing mechanism for SignalGroups is discussed below.

^c ~~NOTE 3~~—This SignalGroup adds four more group definitions to groups previously defined in the unnamed SignalGroup. These definitions contain the same signals, in the same order, but add references about a “Base” to each declaration.

The “base” statement is used to define a default number base to be used for assignment statements referencing this group name. STIL supports the “WFC” base, which is the default mapping of WaveformChars one-to-one to signals in the group; the “hex” base, which uses hexadecimal notation for defining WaveformChar mapping; or the “decimal” base, which uses decimal notation to define WaveformChar references.

To define “hex” or “decimal” mapping, the mapping of WaveformChars to bit values in the hex or decimal number shall ~~must~~ be defined. This definition is provided by WaveformChar references after the hex or decimal word. In this example, the first group defined is ABUS_I. ABUS_I is defined to use a hex base for signal assignment, and the hex values are defined to map to the WaveformChars 0 and 1.

The number of bits in the hex value required to specify the WaveformChar for each signal in a group is determined by the number of the WaveformChar references present in the Base statement. In the definition of ABUS_I, two WaveformChars are referenced. This requires one bit of a hex character to define the WaveformChar reference. The bit value to WaveformChar mapping is performed linearly: the first WaveformChar reference is assigned the bit value 0, the second WaveformChar reference is defined the value 1, and so on. Note that the values increase from left-to-right in this process; the left-most WaveformChar is assigned zero, and each subsequent WaveformChar is incremented. This process may be extended for as many WaveformChar characters [as](#) desired.

It is critical to remember here that the only thing being defined is a relationship of bit-values inside a hex (or decimal) value, to WaveformChars.

The number of bits used for a hex or decimal value is always discrete for each signal in a group. If three WaveformChars are defined in a hex or decimal Base, then two bits are required to define those three states. Unused values of the binary field (such as the value “3” in a three-WaveformChar definition) cannot be specified.

In the third group definition, the group ABUS_O is defined with four WaveformChar references. The mapping of a hex value assigned to this group is demonstrated in [Table 4](#) [Figure 8](#).

[d](#) ~~NOTE 4~~—In this Timing block, the signals ABUS and BBUS are given multiple waveform definitions using two wave-form statements each. Note that a single WaveformChar can only be used once in a WaveformTable per signal. Both ABUS and BBUS are given WaveformChars: 0 and 1 for input waveforms, and H, L, Z, and X for output waveforms.

Note that while the ordered definition of WaveformChars in the WaveformTable matches the order defined in the “base” statements contained in SignalGroups “more,” there is no relationship between these two ordered sets. The order of WaveformChars in WaveformTables is aligned with the events defined in the waveform. The order of WaveformChars in base statements in a SignalGroup defines a value used to map based-numbers to WaveformChars in vector statements.

^e ~~NOTE 5~~—The SignalGroups block named “more” ~~is~~ ~~must be~~ explicitly referenced to be used. The SignalGroups statement here provides for the definitions in this named block to be available to any Patterns referenced here. Also note that the Timing and Pattern names are the same (both are blocks named “basic”). Even though these names are the same, the name space for each block type is different and, therefore, they refer to separate, unique blocks.

^f ~~NOTE 6~~—As stated in the comment above the first V{} statement, the first vector in this Pattern ~~shall~~ ~~must~~ define states for all signals that ~~are~~ ~~will be~~ used in this Pattern because there were no DefaultState values defined for these signals.

This vector uses the group ALL. ALL was defined without a base statement and, therefore, defaults to a one-to-one mapping of WaveformChars to signals in the group.

^g ~~NOTE 7~~—The next vector is an incremental data vector. In STIL, only the data that changes from one vector to the next needs to be identified. This vector makes use of the bus definitions in the SignalGroup “more” even though most of the bits of these busses do not always change. ABUS_I is assigned the value 00 in this second vector, which ~~are~~ ~~will be~~ interpreted as a hex value because of the definition of this group. Hex 00 maps to the bits 00000000. ABUS_I was defined with two WaveformChar references, so each bit of this value is a reference to a WaveformChar value for a signal in this group. All bits of ABUS are assigned the WaveformChar “0.”

BBUS_O is assigned the value 0000, which is again interpreted as a hex value because of the declaration of BBUS_O. This expands to 16 0’s; however, BBUS_O was defined with four WaveformChar references, and so every two bits of this value corresponds to a WaveformChar value for each signal in this bus. Each signal in BBUS_O is assigned the WaveformChar “L” for this statement.

The next eight vectors are essentially the same vectors as in “hello tester,” as the A-bus is being driven and the B-bus is being sampled. The walking-bit pattern is repeated in this sequence.

^h ~~NOTE 8~~—On the 11th vector, the individual signal OE_ is referenced directly. This signal is assigned the WaveformChar “1,” which ~~holds~~ ~~will hold~~ the output disabled for this test cycle. BBUS is ignored in this vector from the “X” state, as all signals are assigned the bit-values “11.”

The vector data then continues, testing the opposite direction of data propagation in this device.

Table 4 ~~Figure 8~~—Mapping of a hex value to the group ABUS_O

Group ABUS_O is defined to map the following WaveformChar references:				
WFC:	L=00	H=01	Z=10	X=11
For example:				
Hex Value	"B"	Bits	" <u>10</u> <u>11</u> "	Z for the first signal ref (value 10)
				X for the second signal (value 11)

5.4 STIL timing expressions/"Spec" information

This example defines a test for the LS245 design using spec timing information. Spec timing parameters are defined using STIL constructs, and waveforms and stimulus are created to test device response against those parameters. This test validates timing against "typical" parameters, which are defined here to be an arbitrary amount less restrictive than the Max values. [The example in Table 5 defines a test for the LS245 design using spec timing information.](#)

Table 5 Figure 9—Spec timing tests LS245

	<pre>STIL 1.0; Signals { DIR In; OE_ In; A0 InOut; A1 InOut; A2 InOut; A3 InOut; A4 InOut; A5 InOut; A6 InOut; A7 InOut; B0 InOut; B1 InOut; B2 InOut; B3 InOut; B4 InOut; B5 InOut; B6 InOut; B7 InOut; } SignalGroups { ABUS = 'A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0'; BBUS = 'B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0'; BUSES= 'ABUS + BBUS'; ALL = 'DIR + OE_ + BUSES'; } SignalGroups more { ABUS_I = 'ABUS' { Base Hex 01; } BBUS_I = 'BBUS' { Base Hex 01; } } a Spec tmode_spec { b Category tmode { tplh { Typ '13.00ns';Max '12.00ns'; } tphl { Typ '13.00ns';Max '12.00ns'; } tpz1 { Typ '41.00ns';Max '40.00ns'; } tpzh { Typ '41.00ns';Max '40.00ns'; } tplz { Typ '26.00ns';Max '25.00ns'; } tphz { Typ '26.00ns';Max '25.00ns'; } strobe_width = '20ns'; tperiod = '500ns'; } } c Selector tmode_typ { tplh Typ;</pre>
--	---

[\(New\)](#)

```

tphl      Typ;
tpz1      Typ;
tpzh      Typ;
tplz      Typ;
tphz      Typ;
}

Timing to_specs {
  WaveformTable pulsed_oe {
    Period 'tperiod';
    Waveforms {
d      DIR { 01 { '0ns' D/U; }}
e      OE_ { 01 { '0ns' U; OE_MARK: '200ns' D/U;
f      OE_CLOSE: 'OE_MARK+100ns' U; }}
      BUSES{ 01 { '10ns' D/U; }
            L { '0ns' Z;'0ns' X; 'OE_MARK+tpz1' l;
              '@+strobe_width' X;}
            H { '0ns' Z;'0ns' X; 'OE_MARK+tpzh' h;
              '@+strobe_width' X;}
            D { '0ns' Z;'0ns' X; 'OE_CLOSE+tplz' t;
              '@+strobe_width' X;}
            U { '0ns' Z;'0ns' X; 'OE_CLOSE+tphz' t;
              '@+strobe_width' X;}
            X { '0ns' Z;'0ns' X; }}
      } // end Waveforms
    } // end WaveformTable pulsed_oe
  WaveformTable const_oe {
    Period 'tperiod';
    Waveforms {
g      DIR { 01 { '0ns' D/U; }}
      OE_ { 01 { '0ns' D; 'tperiod-strobe_width' U;}}
      BUSES{ 01 { IN_MARK: 'tperiod/10' D/U; }
            L { '0ns' Z;'0ns' X; 'IN_MARK+tph1' l;
              '@+strobe_width' X;}
            H { '0ns' Z;'0ns' X; 'IN_MARK+tplh' h;
              '@+strobe_width' X;}
            X { '0ns' Z;'0ns' X; }}
      } // end Waveforms
    } // end WaveformTable const_oe
  } // end Timing to_specs

  PatternBurst spec_check_burst {
    SignalGroups more;
    PatList { spec_check; }
  } //end PatternBurst spec_check_burst

  PatternExec {
h    Timing to_specs;
    Selector tmode_typ;
    Category tmode;
    PatternBurst spec_check_burst;
  } //end PatternExec

  Pattern spec_check {
    W pulsed_oe;
    // the first vector specifies states on all signals.

```

[\(New\)](#)

```

V { ALL=00DDDDDDDDXXXXXXXX; }
// first set of tests check delays from OE_ signal
V { ABUS_I=00;BBUS=LLLLLLLL; } //check BBUS tpz1 spec
V { ABUS_I=FF;BBUS=HHHHHHHH; } //check BBUS tpzh spec
V { ABUS_I=00;BBUS=DDDDDDDD; } //check BBUS tplz spec
V { ABUS_I=FF;BBUS=UUUUUUUU; } //check BBUS tphz spec
V { DIR=1; ABUS=XXXXXXXX; BBUS=DDDDDDDD; }
V { BBUS_I=00;ABUS=LLLLLLLL; } //check ABUS tpz1 spec
V { BBUS_I=FF;ABUS=HHHHHHHH; } //check ABUS tpzh spec
V { BBUS_I=00;ABUS=DDDDDDDD; } //check ABUS tplz spec
V { BBUS_I=FF;ABUS=UUUUUUUU; } //check ABUS tphz spec
W const_oe;
// second set of tests check data propagation delays
V { BBUS_I=00;ABUS=LLLLLLLL; } //check ABUS tph1 spec
V { BBUS_I=FF;ABUS=HHHHHHHH; } //check ABUS tplh spec
V { DIR=0; BBUS=XXXXXXXX; ABUS=XXXXXXXX; }
V { ABUS_I=00;BBUS=LLLLLLLL; }
V { ABUS_I=FF;BBUS=HHHHHHHH; } //check BBUS tplh spec
V { ABUS_I=00;BBUS=LLLLLLLL; } //check BBUS tph1 spec
} //end Pattern spec_check

```

[\(New\)](#)

STIL 1.0;

The numbers in the circles (e.g., ①) correspond to the figure notes that follow.

```
Signals {  
  DIR In;  
  OE_ In;  
  A0 InOut; A1 InOut; A2 InOut; A3 InOut;  
  A4 InOut; A5 InOut; A6 InOut; A7 InOut;  
  B0 InOut; B1 InOut; B2 InOut; B3 InOut;  
  B4 InOut; B5 InOut; B6 InOut; B7 InOut;  
}
```

```
SignalGroups {  
  ABUS = 'A7 + A6 + A5 + A4 + A3 + A2 + A1 + A0';  
  BBUS = 'B7 + B6 + B5 + B4 + B3 + B2 + B1 + B0';  
  BUSES= 'ABUS + BBUS';  
  ALL = 'DIR + OE_ + BUSES';  
}
```

```
SignalGroups more {  
  ABUS_I = 'ABUS' { Base Hex 01; }  
  BBUS_I = 'BBUS' { Base Hex 01; }  
}
```

```
Spec tmode_spec {  
  Category tmode {  
    ①  
    ②  
    tplh { Typ '13.00ns';Max '12.00ns'; }  
    tphl { Typ '13.00ns';Max '12.00ns'; }  
    tpz1 { Typ '41.00ns';Max '40.00ns'; }  
    tpzh { Typ '41.00ns';Max '40.00ns'; }  
    tplz { Typ '26.00ns';Max '25.00ns'; }  
    tphz { Typ '26.00ns';Max '25.00ns'; }  
    strobe_width = '20ns';  
    tperiod = '500ns';  
  }  
}
```

```
Selector tmode_typ {  
  ③  
  tplh Typ;  
  tphl Typ;  
  tpz1 Typ;  
  tpzh Typ;  
  tplz Typ;  
  tphz Typ;  
}
```

(Deleted)