



IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450TM-1999) for Tester Target Specification

IEEE Computer Society

Sponsored by the
Test Technology Standards Committee

IEEE
3 Park Avenue
New York, NY 10016-5997, USA
7 September 2007

IEEE Std 1450.3TM-2007

*Recognized as an
American National Standard (ANSI)*

IEEE 1450.3™-2007

IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450™-1999) for Tester Target Specification

Sponsor

**Test Technology Standards Committee
of the
IEEE Computer Society**

Approved 24 August 2007

American National Standards Institute

Approved 8 March 2007

IEEE SA-Standards Board

Abstract: The STIL environment supports transferring tester-independent test programs to a specific automated testing equipment (ATE) system. Although native STIL data are tester independent, the actual process of mapping the test program onto tester resources may be critical, and it is necessary to be able to completely and unambiguously specify how the STIL programs and patterns are mapped onto the tester resources. TRC (which stands for either tester resource constraints or tester rules checking, depending on the usage) is an extension to the STIL language to facilitate this operation.

Keywords: Tester rules checking (TRC), tester resource reporting, tester resource targeting, tester resource loading

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

Copyright © 2007 by the Institute of Electrical and Electronics Engineers, Inc.
All rights reserved. Published 7 September 2007. 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.

Print: ISBN 0-7381-5517-9 SH95624
PDF: ISBN 0-7381-5518-7 SS95624

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

IEEE Standards documents are developed within the 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 IEEE Standard is wholly voluntary. The IEEE disclaims 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 IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims 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. IEEE Standards documents are supplied “**AS IS**.”

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

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other 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. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

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

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

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.

Introduction

This introduction is not part of IEEE 1450.3-2007, Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450™-1999) for Tester Target Specification.

STIL is a collection of standards with the base standard being 1450 and the dotted extensions used to define additional syntax for addressing additional areas; i.e., this standard addresses tester rules.

The extensions follow the same conventions as the base standard. The base and the extensions are developed so as to work together; i.e., STIL is a single language that is defined (and has been developed) as separate IEEE standards.

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.

Participants

The following is a list of participants in the STIL Working Group.

Tony Taylor, *Chair*

John V. Cosley
David Dowding
Oleg Erlich
Yung D. Fan
Dave Gallagher

Bruce Kaufman
Ken Mandl
Gregory Maston
Gary Murray
Chris J. Nelson

Ken Posse
Paul J. Reuter
Jose M. Santiago
Doug Sprague
Allen Yeats

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

Keith Chow
Tommy P. Cooper
John V. Cosley
Sourav K. Dutta
Yung D. Fan
Randall C. Groves
Kazumi Hatayama
Werner Hoelzl
Chi Tin Hon
Dennis Horwitz
Hirofumi Kamitokusari
Mark J. Knight

Susan K. Land
Adam W. Ley
G. L. Luri
Gregory Maston
Tom Micek
Gary L. Michel
Yinghua Min
Chris J. Nelson
Michael S. Newman
Noriaki Okumiya
Ulrich Pohl

Paul J. Reuter
Robert A. Robinson
Jose M. Santiago
Bartien Sayogo
Roger J. Sowada
Walter Struppler
K. S. Subrahmanyam
Tony Taylor
Srinivasa R. Vemuru
Thomas M. Wandeloski
Gregg Wilder
Oren Yuen

When the IEEE-SA Standards Board approved this standard on 8 March 2007, it had the following membership:

Steve M. Mills, *Chair*
Robert M. Grow, *Vice Chair*
Don Wright, *Past Chair*
Judith Gorman, *Secretary*

Alex Gelman
William R. Goldbach
Arnold M. Greenspan
Joanna N. Guenin
Julian Forster*
Kenneth S. Hanus
William B. Hopf
Richard H. Hulett

Hermann Koch
Joseph L. Koepfinger*
John Kulick
David J. Law
Glenn Parsons
Ronald C. Petersen
Tom A. Prevost

Narayanan Ramachandran
Greg Ratta
Robby Robson
Anne-Marie Sahazizian
Virginia C. Sulzberger
Malcolm V. Thaden
Richard L. Townsend
Howard L. Wolfman

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Richard DeBlasio, *DOE Representative*
Alan H. Cookson, *NIST Representative*

Michelle D. Turner
IEEE Standards Program Manager, Document Development

Michael D. Kipness
IEEE Standards Program Manager, Technical Program Development

Contents

1.	Overview.....	1
1.1	Scope.....	2
1.2	Purpose.....	2
1.3	TRC limitations.....	3
2.	Normative references.....	3
3.	Definitions	3
4.	Structure of this standard	4
4.1	Formats from STIL.0	4
4.2	Additional formatting conventions	5
4.3	Dependencies on IEEE Std 1450.1	5
5.	STIL syntax description.....	6
5.1	Additional reserved words	6
5.2	Keywords used in a TRC block	6
6.	Statement usage and organization by flow	7
6.1	TRC usage for ATE constraint specification.....	8
6.2	TRC usage for design/pattern constraints.....	9
6.3	TRC usage for pattern reporting	9
6.4	TRC usage for tester targetting	10
7.	STIL statement.....	10
7.1	STIL syntax.....	11
7.2	STIL example	11
8.	Variables block extensions	11
8.1	Variables block syntax.....	11
8.2	Variables example.....	12
9.	Resource statement	13
9.1	Resource statement syntax	13
10.	TRC: TestResourceConstraints block.....	14
10.1	TRC syntax	15
10.2	TRC example	18
10.3	TRC block sharing rules	19
11.	TRC: SignalAttributes	19
11.1	TRC: SignalAttributes—syntax	19
11.2	TRC: SignalAttributes—examples	21

12.	TRC: DCResourceAttributes	22
12.1	TRC: DCResourceAttributes—syntax	22
12.2	TRC: DCResourceAttributes—example	24
13.	TRC: PeriodAttributes	25
13.1	TRC: PeriodAttributes—syntax	25
13.2	TRC: PeriodAttributes—examples	26
14.	TRC: WaveformAttributes	26
14.1	TRC: WaveformAttributes—syntax	27
14.2	TRC: WaveformAttributes—examples	31
15.	TRC: WaveformDescriptions	32
15.1	TRC - WaveformDescriptions—syntax	32
15.2	TRC: WaveformDescriptions—examples	34
16.	TRC: PatternAttributes	35
16.1	TRC: PatternAttributes—syntax	35
16.2	TRC: PatternAttributes—examples	38
17.	TRC: NameChecks block	39
17.1	NameChecks block—syntax	39
17.2	NameChecks—examples	40
	Annex A (informative) Glossary	42
	Annex B (informative) Fluid concepts in parameter specification	48
	Annex C (informative) Tester channel map	50
	Annex D (informative) Example of TRC for a simple tester model	52
	Annex E (informative) Example of TRC used to define waveforms and timing	54
	Annex F (informative) Example of TRC used for resource reporting	58
	Annex G (informative) Example of tester targeting and tester loading	61
	Annex H (informative) Example of vector memory checking	66
	Annex I (informative) Waveform generator model	69
	Annex J (informative) File encryption	75
	Annex K (informative) Regular expression reference	76

IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450TM-1999) for Tester Target Specification

1. Overview

The STIL environment supports transferring tester-independent test programs to a specific automated testing equipment (ATE) system. Although native STIL data are tester independent, the actual process of mapping the test program onto tester resources may be critical, and it is necessary to be able to completely and unambiguously specify how the STIL programs and patterns are mapped onto the tester resources. TRC (which stands for either tester resource constraints or tester rules checking, depending on the usage) is an extension to the STIL language to facilitate this operation.

Figure 1 shows the usage model for tester targeting. The four ways that the TRC statements come into play in the flow of data from design to test are indicated by the circled numbers in the diagram. These four uses are defined as follows:

- a) **Tester rules checking:** As early as possible in the process of inserting “Design for Test” logic and generation of test patterns, the rules of the target tester are identified by means of the TRC file defining the target tester.
- b) **Tester resource reporting:** As part of the pattern generation process, a report of resources required for the pattern may be created in TRC format. This information is available for test planning purposes, such as 1) when a pattern is for an embedded core to be integrated into a chip or 2) for tester scheduling purposes. Each resource report is associated with a particular STIL file/stream. The resource report data may be a separate file (as implied in the above diagram) or may be included in the STIL pattern file.
- c) **Tester resource targeting:** The process of tester targeting is that of adding additional information into the STIL file/streams that specifies how the resources of a given tester are to be assigned. Note the bars on the left side of the diagram, which indicate that this targeting operation can be done in one of three places: 1) by the EDA software that generates the patterns, 2) by software created by the test user, or 3) by the ATE software that loads the STIL patterns.
- d) **Tester resource loading:** The tester loader is a process that maps the device-oriented STIL data to the resources of the tester. There may or may not be targeting information provided. If targeting information is not present, then the loader is expected to do the job of assigning resources. If targeting information is present, then it is to be used to direct the resource assignment.

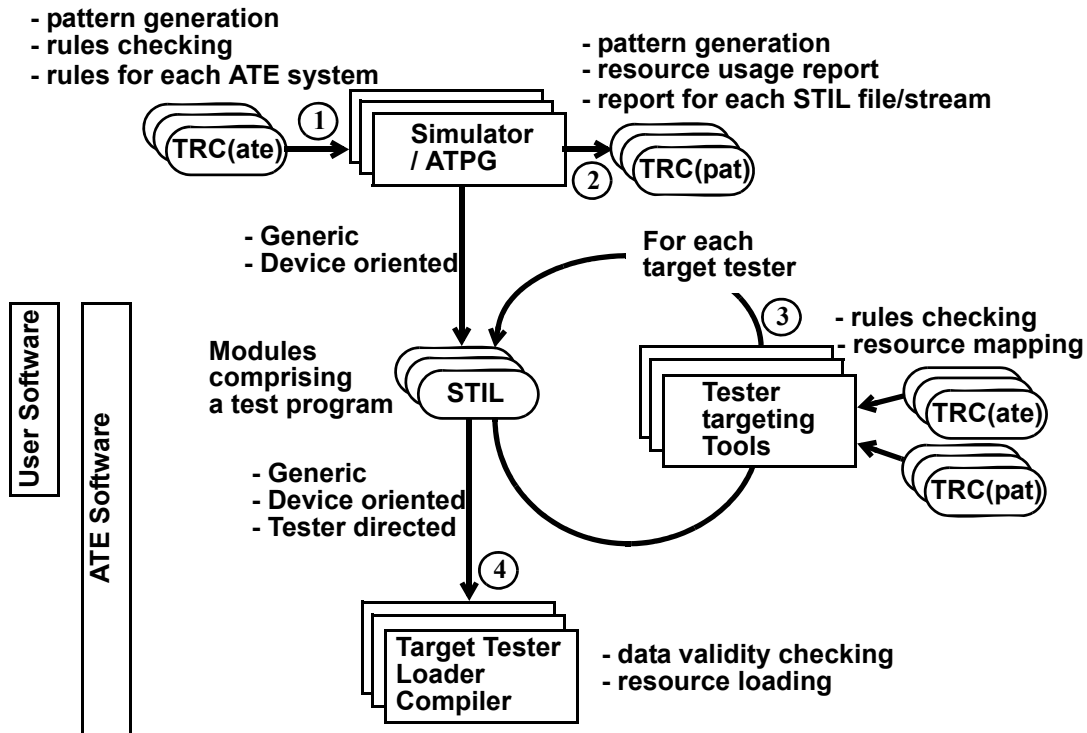


Figure 1—STIL.3 usage model (tester targeting)

Note that the semantics of the various TRC statements change, based on which flow is being addressed; i.e., in a constraint flow, the TRC statements provide a set of rules that are to be applied to a set of data and an error shall be generated if the test pattern data does not conform to the rules; in a report flow, the same TRC statements are providing a summary of test pattern data regardless of whether it conforms to any given TRC constraints. Most TRC statements can be used in either context. The definitions in this standard are written from the constraint point of view.

1.1 Scope

- Define structures in STIL for the specification of resource mapping of ATE hardware architectures. An example of resource mapping is the assignment of tester resources to waveform characters that are used in STIL vectors.
- Define structures in STIL for including ATE-specific instructions in-line with the STIL data.
- Define structures in STIL that allow for “incremental processing” whereby, a set of STIL files may be targeted to multiple ATE systems by allowing separately identified ATE data to coexist.
- Define structures in STIL for defining tester rules checks to ensure that the set of generated STIL files conform to the selected resources on one or more ATE systems.
- Define structures in STIL for the specification of the resources required for the execution of a set of STIL files on a given ATE system.

1.2 Purpose

Transferring “tester independent” test program/pattern data as represented in STIL to a specific ATE system is a desired capability. It is required to be able to completely and unambiguously specify how the STIL program/patterns are mapped onto a specific tester’s resources. Because of the various different use models for the creation and consumption of test data, it is necessary to enable certain operations (such as rules

checking) very early in the process. Likewise it is desirable to allow other operations (such as resource allocation) to be done very late in the process. The STIL language extensions enable the user/creator a standard way of specifying and controlling the application of test program/pattern data to specific ATE systems to the extent necessary for each use model scenario.

1.3 TRC limitations

In setting the scope for any standard, some issues are identified and determined not to be defined. The following is a list of issues that are not in the scope of this standard:

- a) Violations: The tester-targeting operation naturally leads to situations where a given set of constraints cannot be met. It is not in the scope of this standard to define the content or format of such constraint rule violations.
- b) Completeness: The variations in architecture of various ATE systems makes it virtually impossible to address all the nuances and corner cases of each design. It is the intent of this standard to provide a common way of describing key common attributes. This standard is intended to address 80% of the issues involved with TRC operations, because the effort to address the remaining 20% is too complex and would make the standard unwieldy to apply in general.

2. Normative references

The following referenced documents are indispensable for the application of this Standard. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1450TM (STIL.0), IEEE Standard Test Interface Language (STIL) for Digital Test Vectors.^{1, 2}

IEEE Std 1450.1TM (STIL.1), IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450TM-1999) for Semiconductor Design Environments.

IEEE Std 1450.2TM (STIL.2), IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std 1450TM-1999) for DC Level Specification.

IEEE Std 1450.6TM (STIL.6), IEEE Standard IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data-Core Test Language (CTL).

3. Definitions

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

STIL.0: Refers to IEEE Std. 1450. This base STIL standard is commonly referred to as “dot 0”.

¹The IEEE standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.

²IEEE publications are available from the Institute of Electrical and Electronics Engineers, Inc., 445 Hoes Lane, Piscataway, NJ 08854, USA (<http://standards.ieee.org/>).

STIL.1: Refers to IEEE Std 1450.1. This extension to the STIL base standard is commonly referred to as “dot 1”: Design information specification.

STIL.3: Refers to IEEE Std 1450.3 (i.e., this standard). This extension to the STIL base standard is commonly referred to as “dot 3”: Tester target specification.

STIL.6: Refers to IEEE Std 1450.6. This extension to the STIL base standard is commonly referred to as “dot 6”: Core test language specification.

TRC: (A) (Tester resource constraint) a set of STIL statements (i.e., this standard) that are used to specify either 1) the resources available in a tester or 2) the resources needed in support of a STIL pattern. (B) (Tester rule check) an operation performed by an application to verify the consistency between a pattern and a set of tester resource constraints.

4. Structure of this standard

This document is an adjunct to IEEE Std 1450. The conventions established and defined in IEEE Std 1450 are used in this document and are included verbatim below.

Many clauses in this document add additional constructs to existing clauses in the IEEE Std 1450 document and are so identified in the title. The constructs defined in this document are limited to the Environment block as defined by IEEE Std 1450.1. All clauses in this document are normative unless specifically identified as “Informative.” Example code is provided within each clause. More complete examples are provided in the annexes, which are informative.

4.1 Formats from STIL.0

The following discussion is a copy of the conventions as defined in STIL.0 and followed by this document.

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:

- 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 that may be used 0 or 1 time.
- e) ()+ indicates syntax that may be used 1 or more time.
- f) ()* indicates optional syntax that 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:

- a) 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 to issue an error if the requirement is not met by the input.
- b) To developers of STIL file/stream (writers), “shall” denotes mandatory characteristics of the language. The resulting output must conform to these characteristics.
- c) 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 list above.

4.2 Additional formatting conventions

The following items are new conventions that are used in the various paragraph types:

- a) In the Syntax definitions, each statement or group of statements is identified by a syntax line number (in parenthesis at the right side of the page). These numbers are to be referenced to the definitions that follow, which contain the syntax line numbers in parenthesis on the left side of the page.
- b) In the Syntax definitions, underlined attribute keywords indicate the default when none in a group is selected.
- c) In the code example sections of the document, the text is in `Courier` font and each line contains a line number followed by a colon (:) at the left-hand side of the page. This line number is for reference only and is not part of the code.
- d) In the code examples, user-defined names (for things like signalnames, signal group names, block names, etc.) are in all uppercase with under_score separators. This is to distinguish them from STIL keywords.
- e) In the tutorial examples (Annex D through Annex G), circled numbers on the right-hand side of the code examples are used to indicate that a note about the code is included immediately after the example.

4.3 Dependencies on IEEE Std 1450.1

This standard is built on IEEE Std 1450 and relies on some of the syntax as defined in the Design extension (IEEE Std 1450.1). The following list provides the IEEE Std 1450.1 statements that are used by this standard:

- **Clause 5, Expressions constructs:** The additional expression capabilities are used for defining relationships between various TRC attributes (see Annex B).
- **Clause 7, STIL statement:** This statement, which is required at the beginning of every STIL file, is extended to allow specification of the STIL extension standards.
- **Clause 9, Variables block:** This new block is required when fluid attributes are used (see Annex B).
- **Clause 18: Environment block:** The block allows for the definition of STIL information pertaining to the environment of application in which a set of test pattern data is used. TRC is one such environment, and hence, all TRC statements are contained within an Environment block.
- **18.2 “MAP_STRING” syntax and 18.3 NameMaps example:** As part of the general Environment block, syntax is defined that defines naming relationships to other environments. Tester channel information makes use of this facility (see Annex C).

5. STIL syntax description

This clause contains extensions to Clause 6 of STIL.0. All constructs and restrictions for Clause 6 of IEEE Std 1450 are in effect here, with the following additions.

5.1 Additional reserved words

Table 1 lists new STIL reserved words defined by this standard and not defined in IEEE Std 1450. Subsequent clauses in this standard identify the use and context of each of these additional reserved words.

Table 1—Additions to STIL reserved words

Assert
ConfigConstant
ParamConstant
Resource

5.2 Keywords used in a TRC block

Table 2 lists new STIL keywords that are used with a TRC block or TRC sub-block. These keywords are not reserved in the general STIL context, but they are available only inside a TRC block or a TRC sub-block.

Table 2—TRC keywords

Accuracy, AllowedWhen, AppliesTo
Base, Block
CharacterContent, CompareEvents, ConditionalStatements, Contents, CoreUsageReady
DeltaChangeVectorData, DriveEvents
AllowedScanPadWaveforms
DCLimits, DCResourceAttributes, DifferentialConfiguration
Environment
FanOut, FormatSelect
Infinite, InOut, InstructionAttributes
Length, LoopAttributes
Macro,
Max, MaxCaptureMemory, MaxData, MaxEdgeTime, MaxIO, MaxMask, MaxPeriodGenerators, MaxPeriods, MaxShapes, MaxScanChainLength, MaxIteration, MaxLength, MaxNest, MaxRunTime, MaxScanMemory, Max-Signals, MaxTimeSets, MaxTimingGenerators, MaxVectorMemory,
MinAfter, MinBefore, MinCompareWindow, MinCompareToDriveOn, MinDriveOffTime, MinDriveOffToCompare, MinDriveOnTime, MinDrivePulse, MinEdgeReTrigger, MinIteration, MinLength, MinTimeAfterMatch, Min-VectorsAfter, MinVectorsAfterMatch, MinVectorsBefore, MinVectorsBetween,
Modulus, MultiBitData, MultipleDevices, MultipleSites

Table 2—TRC keywords (continued)

NameChecks, NonCyclized, NumberCaptureCycles, NumberData, NumberDCLevels, NumberIO, NumberLevels, NumberMask, NumberPatternUnits, NumberPeriods, NumberShapes, NumberSignals, NumberTesterChannels, NumberTimeSets, NumberVectorsPerShift
PatternAttributes, PatternVariables, PeriodAttributes, PeriodSelectMemory, PerPinConfiguration, PerTimingGenerator, Pragma, ProcedureCalls
Resolution
Scope, SelectWithPeriod, Shape, SignalAttributes, SignalsPer, STILPatterns, SubWaveformDuration, SubWaveformIteration, SystemAttributes
TimeLimits
TRC
UndrivenInOut, Units, Usage
VectorCompression
WaveformDescriptions, WaveformAttributes, WaveformSelectMemory

6. Statement usage and organization by flow

As shown in Figure 1, there are four ways in which TRC blocks and statements can be utilized to accomplish various objectives. Table 3 shows the blocks of data that are pertinent to each of these four usage models for TRC data.

There are no special requirements for sequencing of blocks within a TRC block. A reader is expected to process an entire TRC block prior to interpreting.

A TRC block that contains a PatternReport is typically part of the STIL file/stream for that pattern (or a referenced Include file). The typical placement of this block (or Include) would be near the beginning, right after the definition of Signals, Groups, Specs, and Variables.

A TRC block that contains Constraint data is typically a separate STIL file/stream from the STIL file/stream that contains the pattern data. The constraint data file and the pattern data file would be separately identified by the tool or ATE software that it processing the STIL file/stream(s).

Table 3—STIL/TRC block usage

Block/Statement/Function	Purpose	Constraint ^a	Report ^b	Target ^c	Load ^d
STIL	file header id	X	X	X	X
Environment { }	container block	X	X		X
Environment { TRC { Usage Constraints; } }	specify to use for constraints	X			X
Environment { TRC { Usage PatternReport; } }	specify that data is a report		X		
Environment { TRC { Category { } }	select one or more categories	X			X
Environment { TRC { SystemAttributes { } }	define system application	X			X
Environment { TRC { NameChecks { } }	define name constraints	X			
Environment { TRC { PatternAttributes { } }	define pattern information	X	X		X
Environment { TRC { PeriodAttributes { } }	define period information	X	X		X
Environment { TRC { SignalAttributes { } }	define signal information	X	X		X
Environment { TRC { WaveformAttributes { } }	define waveform information	X	X		X
Environment { TRC { WaveformDescriptions { } }	define wave shapes	X	X		X
STIL blocks	any block in STIL			X	X
Environment { NameMaps { } }	map STIL names to other form			X	X
Pragma { }	define ATE native statements			X	X
<<resource_id>> within STIL blocks	define ATE resource mapping			X	X

^aConstraint = Information that is used to convey constraints for an ATE system, or else a set of arbitrary instructions to be used to constrain the implementation of a chip design or chip test patterns.

^bReport = Information that is used to report the actual parameters of a given pattern or pattern burst.

^cTester Target = Information that is added to a STIL file/stream to specify how it is to be loaded into the resources of a given ATE system.

^dTester Loading = Information that is used in a STIL file/stream to control the loading of the resources of an ATE system.

6.1 TRC usage for ATE constraint specification

An ATE specification file is typically a separate file that specifies the attributes of one or more ATE systems. It may contain a set of attributes that is the intersection of several ATE systems, thereby ensuring that a pattern that meets the constraints will run on any ATE system. The file may define a class of ATE systems by utilizing category variables or expressions. The structure of an ATE constraint file is as follows:

```

1: STIL 1.0 { Design 2005; TRC 2007; }
2: Header {
3:   Source "IEEE Std 1450.3-2007";
4:   Ann { * clause 6.1 * }
5: }
6: Environment {

```

```

7:   TRC ATE_1 {
8:     Usage Constraints;
9:   }
10:  TRC ATE_2 {
11:    Usage Constraints;
12:  }
13:} //end Environment

```

6.2 TRC usage for design/pattern constraints

A TRC file that is to be used for constraining the creation of a design or pattern set for that design looks much the same as the ATE constraint file. The difference is that typically only one set of constraints will exist and hence only one Environment block in the file. If there are multiple Environment blocks, then the name of the block is an identifier for a set of constraints that are selected by the design tool. The structure of a file with a single set of constraints is as follows:

```

14:STIL 1.0 { Design 2005; TRC 2007; }
15:Header {
16:  Source "IEEE Std 1450.3-2007";
17:  Ann { * clause 6.2 * }
18:}
19:Environment {
20:  TRC {
21:    Usage Constraints;
22:  }
23:} //end Environment

```

6.3 TRC usage for pattern reporting

The same syntax as is used for specifying resource constraints on a Pattern (as defined in STIL.0) can also be used to report the resource utilization for that Pattern (or PatternBurst). The different application is identified as such by the “Usage PatternReport;” statement in the TRC block. Some resource constraint blocks do not apply in this application, i.e., System and NameChecks. The information could be included in the same STIL file as the pattern or in a separate file. The structure of this type of file is as follows:

```

24:STIL 1.0 { Design 2005; TRC 2007; }
25:Header {
26:  Source "IEEE Std 1450.3-2007";
27:  Ann { * clause 6.3 * }
28:}
29:Environment {
30:  TRC PATNAME_1 {
31:    Usage PatternReport;
32:  }
33:} //end Environment
34:Environment {
35:  TRC PATNAME_2 {
36:    Usage PatternReport;
37:  }
38:} //end Environment

```

6.4 TRC usage for tester targetting

Tester targetting is accomplished by annotating a STIL file/stream with additional information that tells an ATE loader/translator how to map the STIL constructs onto the hardware resources of the tester. This information can take many forms, but the following example shows one possibility:

```
39:STIL 1.0 { Design 2005; TRC 2007; }
40:Header {
41:  Source "IEEE Std 1450.3-2007";
42:  Ann { * clause 6.4 * }
43:}
44:Signals { SIG[1..5] InOut; }
45:Environment ATE1 {
46:  NameMaps { } //specify signal to channel mapping
47:}
48:SignalGroups { SIGS = SIG[1..5]; }
49:Timing basic {
50:  WaveformTable ONE {
51:    <<PER1>> Period 500ns; //tag the period resource
52:    DIR { <<SEQ1>> 01 { 0ns D/U; } } //tag the per pin waveform resource
53:  }
54:  WaveformTable TWO {
55:    <<PER1>> Period 500ns; //use same resource as wft one
56:    DIR { <<SEQ2>> 01 { 0ns D/U; } } //use different resource from wft one
57:  }
58:Pragma ATE1 { *
59:  MAP1: ...
60:  MAP2: ...
61:*}
62:Pattern P {
63:  <<MAP1>> V { SIGS = 10101; } //select MAP1 from pragma ATE1
64:  <<MAP2>> V { SIGS = LHLHL; } //select MAP2 from pragma ATE1
65:  <<MAP1>> V { SIGS = 01010; } //select MAP1 from pragma ATE1
66:}
```

7. STIL statement

This clause contains extensions to Clause 8 of STIL.0.

The STIL statement identifies the primary version of IEEE Std 1450 information contained in a STIL file and the presence of one or more standard Extension constructs. The primary version of STIL is defined in IEEE Std 1450.

The extension to the STIL statement allows for a block containing extension identifiers that allow for additional constructs in the STIL file. Multiple Extension statements may be 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.

All other constructs and restrictions for Clause 8 of IEEE Std 1450 are in effect here.

7.1 STIL syntax

```
STIL IEEE_1450_0_IDENTIFIER { (1)
    ( TRC 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 IEEE Std 1450.

(2) **TRC**: The specific name of this Extension.

EXT_VERSION: The version of this extension to STIL. This standard is identified by the identifier 2006.

7.2 STIL example

```
67:STIL 1.0 { Design 2005; TRC 2007; }
68:Header {
69:  Source "IEEE Std 1450.3-2007";
70:  Ann { * clause 7.2 * }
71:}
```

8. Variables block extensions

This clause defines extensions to Clause 9, STIL.1.

This clause defines additional statements to the Variables block to support fluid constraint definitions. All statements and capabilities as defined in STIL.1 are unchanged.

Fluid constraints are used in this standard to define how a set amount of a resource may be allocated or shared between multiple uses of that resource. Each use may require different amounts of that resource, and there is an identified limit to the amount of that resource available. Fluid constraints may be used to specify aspects of a tester that are optional (i.e., present or not present) or have varying size options (e.g., memory size, or number of channels). Fluid constraints may be used to specify tradeoffs that may be made at load time to support the specific needs of the application. For example, a tester may be able to use two slow-speed pin channels to make one high-speed pin channel; a tester may be able to trade off vector memory for scan memory. See Annex B for an example of the application of this capability.

This standard does not specify how the tradeoff computation is to be accomplished or what might be the consequences of different tradeoffs. However, the Assert statement (defined below) allows for the checking of the fluid assignments to ensure that they satisfy the requirements of the target test system.

8.1 Variables block syntax

```
Variables (VARIABLES_DOMAIN) {
    Assert boolean_expr ; (1)
    ConfigConstant CONFIG_NAME (ALT_CONFIG_NAME)* ; (2)
    ParamConstant PARAM_NAME ; (3)
}
```

(1) **Assert:** The Assert statement contains a boolean expression that is expected to be TRUE in order for the rules checking to pass successfully. As defined in 5.6 of STIL.1, a boolean expression can be one that contains a boolean operator (e.g., $=$, $:=$, or $<>$) or it can be an expression that results in an integer value that is interpreted as a boolean result (i.e., one or greater for TRUE or zero or less for FALSE). If the result of the expression evaluation is FALSE, the application shall report this failure to satisfy this statement. The operands of the boolean expression can be as follows:

- a) Named constants that are defined in the current Variables block: IntegerConstant, ConfigConstant, ParamConstant
- b) Named constants that are defined in a global (unnamed) Variables block
- c) Literal integer values

(2) **ConfigConstant:** This statement defines the name of a value that is determined by the “configuration” of the hardware in a tester. The actual value is to be provided to the TRC process based on knowledge of the target test system. Once defined, the value is then used as a constant value. The constant can be of type integer (see 5.7 of STIL.1 for the definition of allowed integer forms) or real number (see 5.9 of STIL.1 for the definition of allowed real expression forms).

CONFIG_NAME: The name of the hardware configuration parameter. This name may be used in expressions as a defined constant.

ALT_CONFIG_NAME: Zero or more ALT_CONFIG_NAME identifiers may be specified. These names shall reference CONFIG_NAME names that are defined in other ConfigConstant statements that are in scope. The name specifies another resource that is available and capable of satisfying the same requirements and may be used when the CONFIG_NAME resource is exhausted.

(3) **ParamConstant:** This statement defines the name of a value that is assignable by the TRC process. The value chosen is typically a tradeoff with other ParamConstant values and allows the TRC process to optimize the use of resources for a particular device test program. Once defined, the value is then used as a constant value. The constant can be of type integer (see 5.7 of STIL.1 for the definition of allowed integer forms) or a real number (see 5.9 of STIL.1 for the definition of allowed real expression forms).

8.2 Variables example

```
72:STIL 1.0 { Design 2005; TRC 2007; }
73:Header {
74:  Source "IEEE Std 1450.3-2007";
75:  Ann { * clause 8.2 * }
76:}
77:Variables {
78:  // the following illustrates usage of the Assert statement
79:  IntegerConstant K1 := 99;
80:  IntegerConstant K2 := 100;
81:  Assert K1 <> K2;
82:  Assert K1 + K2 <= 200;
83:  Assert K1;
84:  Assert K1 := K2; // FALSE -- report as ERROR
85:
86:  // the following illustrates usage of the ConfigConstant and IntegerParm statements
87:  ConfigConstant NUM_MODULES;
88:  Assert 0 <= NUM_MODULES <= 39;
89:  IntegerConstant NUM_SIGNAL_SLOTS := NUM_MODULES * 32;
90:
91:  ConfigConstant NUM_PATTERN_MEMORY_BITS;
```

```

92: Assert 0 < NUM_PATTERN_MEMORY_BITS <= 4*1024*1024*1024;
93:
94: ParamConstant NUM_800MBPS;
95: ParamConstant NUM_400MBPS;
96: ParamConstant NUM_SCANIN;
97: ParamConstant NUM_SCANOUT;
98:
99: IntegerConstant NUM_SIGNALS :=
100:   NUM_800MBPS + NUM_400MBPS + NUM_ScanIn + NUM_ScanOut;
101: Assert 2*NUM_800MBPS + NUM_400MBPS + NUM_ScanIn + NUM_ScanOut
102:   <= NUM_SIGNAL_SLOTS;
103:
104: ParamConstant NUM_SCAN_LOAD_UNLOAD;
105: ParamConstant LONGEST_SCAN_SHIFT;
106: IntegerConstant SCAN_BITS_NEEDED :=
107:   NUM_SCAN_LOAD_UNLOAD
108:   * LONGEST_SCAN_SHIFT
109:   * (NUM_SCANIN + 2*NUM_SCANOUT);
110:
111: ParamConstant NUM_PATTERN_VECTORS;
112: IntegerConstant MAIN_MEM_NEEDED :=
113:   NUM_PATTERN_VECTORS * NUM_SIGNALS;
114:
115: Assert MAIN_MEM_NEEDED + SCAN_BITS_NEEDED <=
116:   NUM_PATTERN_MEMORY_BITS;
117: }

```

9. Resource statement

The resource statement is used to identify on a STIL statement or block basis, the specific resources within a target tester that are to be assigned for this statement or block. This information is optional and may be computed by the software load operation on the tester (see Figure 1). The content of the resource memory itself may be defined in a Pragma block in a format appropriate to the resource for the particular tester. See Annex G for an example of the usage of this facility.

9.1 Resource statement syntax

Resource (TESTER_IDENTIFIER)+ ; *// list of target testers* (1)
 << (RESOURCE_ID)+ >> (2)

(1) **Resource:** The Resource statement is used to define a list of tester names that are to have resource identifiers. The ordering of the tester_identifiers determines the ordering of the resource_identifier tags in the <<RESOURCE_ID>> tags that follow. Typically this statement will be at the beginning of a pattern block. It shall occur prior to the occurrence of any <<RESOURCE_ID>> tag. The Resource statement and the <<RESOURCE_ID>> tags may be used in either a Pattern or a Timing block, and only one Resource statement shall be allowed within a Pattern or Timing block.

(2) <<RESOURCE_ID>>: The resource id tag is a string contained in double angle brackets that may optionally be placed prior to any block or statement keyword. The purpose of this <<RESOURCE_ID>> tag is twofold: (a) it is used to identify the tester resource that is to be assigned to support the tagged block or statement, and (b) as a reference to some other block (quite possibly a Pragma block) that defines the loading of the tester resource. The surrounding angle brackets are a required part of the syntax and are

expected to be ignored by parsers that are not involved with loading patterns to a tester. There may be multiple RESOURCE_ID names within the angle brackets, separated by a space; in which case, they are to be correlated with the tester names as listed in a Resource statement that shall be contained within the Timing or Pattern block. If no tag exists in a list of resource id tags, this shall be indicated by the use of the asterisk (*) character.

The resource id tag within the angle bracket is a format as required by the tester loader. The following are examples of allowed formats for resource id tags. The actual format that is used for any given tester loader is as defined by that application and is not defined in this standard:

- a) A user-defined name (see definition of user-defined name characteristics in 6.8 of STIL.0)
- b) An integer
- c) An asterisk (*) indicating a null resource id tag

10. TRC: TestResourceConstraints block

The TRC block is used to define the constraints for a given test system or the required resource attributes for a given test program. The TRC block always exists within an Environment block. If a single set of constraints is being defined in a STIL file/stream, then neither the Environment nor the TRC block need be given a name. If the STIL file/stream contains TRC blocks for different classes of tester or different classes of user, then these would typically be in separately named Environment blocks. If the STIL file/stream contains TRC blocks for similar testers (e.g., different models or configurations), then these would typically be in separately named TRC blocks. When a TRC file contains multiple TRC blocks, the application processing that file may process all blocks present or specify which blocks to apply, depending on the application. When a TRC file contains multiple TRC blocks, the application processing that file may process all blocks present or specify which blocks to apply, depending on the application.

Either the Module block or the SignalAttributes block shall serve as the entry point into a set of TRC rules. If all resources are uniform, then a global (unnamed) SignalAttributes block should be used. If resources with different attributes are to be defined, then multiple Modules blocks are used. Refer to the statement syntax definitions in this clause for detailed definition of the rules.

A series of sub-blocks within a TRC block contain attributes of various different aspects of a system: SignalAttributes, PeriodAttributes, or PatternAttributes. Each of these blocks is referenced by either a Module block statement or a Signal block statement to construct the set of constraints that apply to a given instance.

There are facilities in the syntax to allow for sharing of common blocks. Refer to 10.3.

The following are general rules of interpretation that apply to all statements in a TRC block and all blocks that are contained within a TRC block:

- a) In the syntax definitions, optional statements are identified with parenthesis (). When one optional statement is omitted in a STIL-TRC file, then this means that no checking of this constraint shall be done.
- b) An integer value of “-1” means that the parameter is unbounded and therefore need not be checked. This usage is effectively the same as omitting the statement altogether, however, it indicates a purposeful intent rather than a possible oversight.

10.1 TRC syntax

prefix = < E P T G M k m u n p f a > // used by Units statement

eng_unit = < A C e l F H H z m O h m s W V > // used by Units statement

```

Environment (ENV_NAME) { (1)
  ( TRC (TRC_NAME) { (2)
    ( STILExtensions ( < Design | DCLevels | TRC | CTL > )+ ; ) (3)
    ( Usage < Constraints | PatternReport > ; ) (4)
    ( Category (CAT_NAME)+ ; ) (5)
    ( Selector (SEL_NAME)+ ; ) (6)
    ( Variables (VAR_NAME)+ ; ) (7)
    ( MultipleDevices integer_expr; ) (8)
    ( MultipleSites integer_expr; ) (9)
    ( Module (MODULE_NAME) { (10)
      ( DCResourceAttributes DC_ATTR_NAME; ) (11)
      ( MaxNumberModules integer_expr; ) (12)
      ( PatternAttributes (PAT_ATTR_NAME)+ ; ) (13)
      ( PeriodAttributes PER_ATTR_NAME ( < Synchronous | Asynchronous > ) ; ) (14)
      ( SignalAttributes (SIG_ATTR_NAME) +; ) (15)
      ( WaveformAttributes WAV_ATTR_NAME ; ) (16)
      ( WaveformDescriptions WAV_DESC_NAME ; ) (17)
    } ) * // end Module
    ( Units ( < (prefix) eng_unit | Integer | Real > )+ ; ) (18)
    ( dc_resource_attributes_block ) * (19)
    ( name_checks_block ) * (20)
    ( pattern_attributes_block ) * (21)
    ( period_attributes_block ) * (22)
    ( signal_attributes_block ) * (23)
    ( waveform_attributes_block ) * (24)
    ( waveform_descriptions_block ) * (25)
  } ) * // end TRC
} // end Environment

```

(1) **Environment**: The Environment block is a general-purpose construct defined in STIL.1 for the purpose of containing application environments for STIL data. In this case, the Environment block is used to contain tester resource constraint data that are used for transferring data from an EDA (pattern generation) environment into an ATE (pattern consumption environment). The environment block (and hence the environment block name) is used to (a) differentiate from all other environment blocks that may be used for other purposes and (b) to identify classes of TRC blocks (e.g., all like testers from a given manufacturer, or all testers on a test floor).

(2) **TRC**: This statement begins a block describing a set of constraints for a given tester or the set of test requirements for a given test program. The TRC_NAME is typically the name/model of the ATE system that is being defined. If the TRC block is unnamed, then all blocks within it are global (i.e., available within all the named TRC blocks within the containing Environment block). This facility allows for the sharing of common attributes by placing them in the “global TRC block” and referencing them from individual, named, TRC blocks. It is an error to have a block of the same name in both the global and a named TRC block. In the case where multiple blocks are allowed (e.g., Module or PatternAttributes), it is permissible to reference blocks in both the global and the named TRC as long as the names are unique.

(3) **STILExtensions**: This statement is used to indicate which of the STIL extensions are supported by the target tester or test program. If this statement is not present, then only STIL.0 is supported. The extensions available at the time of this writing are **Design**, **DCLevels**, and **CTL**.

(4) **Usage:** This statement is used to indicate the intent of the TRC block. The default if this statement is not present is Constraints. The allowed identifiers are as follows:

Constraints: This keyword specifies that the TRC block is used to define a set of target constraint attributes. These attributes will typically be for a particular model or instance of an ATE system. However, they may also be attributes that a user defines to constrain a test data flow, or constraints that a chip design application applies to a design flow.

PatternReport: Specify that the TRC block contains a report summary of the resources and requirements needed in support of execution of a pattern (or pattern burst).

(5) **Category** (CAT_NAME)+ : This statement is optional and is used to define the category names that are used within the TRC block. The typical usage for variables within TRC is for “fluid” constraints, i.e., constraints that are coupled one to another. See Annex B for information about fluid constraints.

(6) **Selector** (SEL_NAME)+ : This statement is optional and is used to choose a selector block that picks the min, typ, max values for spec-variables. If a selector block is not referenced, then all spec.-variables use the “Typical” values.

(7) **Variables** (VAR_NAME)+ : This statement is optional and is used to choose a variables block. If a named variables block is referenced, the names shall not conflict with the names defined in the global variables block (if one exists). The variables in the global block and the variables in the referenced block(s) are available in the current TRC block.

(8) **MultipleDevices:** This statement defines the number of like devices that may be tested simultaneously in a given tester. If this statement is omitted, then only single device testing is allowed.

(9) **MultipleSites:** This statement defines the number of un-like devices that may be tested simultaneously in a given tester. If this statement is omitted, then only single device type testing is allowed.

(10) **Module:** This statement begins a block that contains attribute information for one module or instrument of a tester. The allowed statements within a Module are defined immediately below. The name of a Module block is optional and is for documentation purpose only (a module is never referenced). There may be multiple unnamed and named Module blocks that together define a system. If an unnamed SignalAttributes exists, then it is an error to have any Module blocks.

(11) **DCResourceAttributes:** This statement references the block that defines the dc resource attributes for the module. There shall be only one DCResourceAttributes statement in the Module block. This statement in the Module block takes precedence over a global, unnamed block.

(12) **MaxNumberModules:** This statement specifies the maximum number of like instances of the module that are allowed. If this statement is omitted, the default value is 1.

(13) **PatternAttributes:** This statement references the block that defines the pattern attributes for the module. There shall be only one PatternAttributes reference statement in a Module block. There may be multiple names in the statement indicating that any one of the referenced blocks may be used; i.e., this is an “OR” function indicating that one OR the other of the named blocks may satisfy the rule checking. “ANDing” of rules sets that are contained in multiple blocks can be accomplished using the Inherit statement. The following code example indicates how the desired block is accomplished:

```
TRC {
    PatternAttributes PA1 { }
    PatternAttributes PA2 { }
}
TRC TRC2 {
    PatternAttributes PA3 { }
```