

# P1450.3

## IEEE Standard Test Interface Language (STIL)

### Extension for Tester Resource Constraints (TRC)

Prepared by the STIL Working Group of the Test Technology Standards Committee

Copyright © <current year> by the Institute of Electrical and Electronics Engineers, Inc.  
 Three Park Avenue  
 New York, New York 10016-5997, USA  
 All rights reserved.

This document is an unapproved draft of a proposed IEEE-SA Standard - USE AT YOUR OWN RISK. As such, this document is subject to change. Permission is hereby granted for IEEE Standards Committee participants to reproduce this document for purposes of IEEE standardization activities only. Prior to submitting this document to another standard development organization for standardization activities, permission must first be obtained from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. Other entities seeking permission to reproduce portions of this document must obtain the appropriate license from the Manager, Standards Licensing and Contracts, IEEE Standards Activities Department. The IEEE is the sole entity that may authorize the use of IEEE owned trademarks, certification marks, or other designations that may indicate compliance with the materials contained herein.

IEEE Standards Activities Department  
 Standards Licensing and Contracts  
 445 Hoes Lane, P.O. Box 1331  
 Piscataway, NJ 08855-1331, USA

Table 1: History

Date	By	Comments
Feb 03, 2002 -D04	Tony Taylor	New draft document. See the review resolution document for details.
Jan 27, 2002 -D03	Tony Taylor	<ul style="list-style-type: none"> <li>- Moved Name Checks to top level of TRC block</li> <li>- WFCMap was removed from the document</li> <li>- The blocks were moved such that all TRC blocks are together and so noted by the titles.</li> <li>- Added an informational block on usage of NameMaps for tester channel mapping.</li> <li>- Copied the scope and purpose from the PAR into this document.</li> <li>- Added full syntax description blocks and removed the in-line descriptions.</li> <li>- Moved Pragma to top level of STIL block</li> <li>- Full list of changes contained in D03 review resolution document</li> </ul>
Nov 10, 2001 -D02	Tony Taylor	<ul style="list-style-type: none"> <li>- Annex A and B, provided by Dan Fan.</li> <li>- Updates to overview fig 1.</li> </ul>
Nov 8, 2001 -D01	Tony Taylor	- New document created. Initial TesterResourceConstraint is moved out of 1450.6-D16 document.

## Table of Contents

<b>1.Overview .....</b>	<b>3</b>
<b>2. References.....</b>	<b>4</b>
<b>3. Definitions, acronyms, and abbreviations .....</b>	<b>4</b>
<b>4. Structure of this standard .....</b>	<b>5</b>
<b>5. Tester targeting orientation and capabilities tutorial (informative).....</b>	<b>6</b>
<b>6. STIL Syntax Description” - Extensions to STIL0 Clause 6.....</b>	<b>6</b>
<b>7. STIL statement - Extensions to STIL0 clause 8 .....</b>	<b>7</b>
<b>8. Pattern -&gt; Resource Statement - Extension to STIL1, Clause 22.....</b>	<b>7</b>
<b>9. Pragma Block .....</b>	<b>9</b>
<b>10.TesterChannelMap - Usage of STIL1, Clause 18.....</b>	<b>9</b>
<b>11.WFCMap Block .....</b>	<b>10</b>
<b>12.TRC - TestResourceConstraints Block.....</b>	<b>10</b>
<b>14.TRC - SignalCharacteristics .....</b>	<b>12</b>
<b>15.TRC - Signals Supply .....</b>	<b>13</b>
<b>16.TRC - PeriodCharacteristics .....</b>	<b>13</b>
<b>17.TRC - WaveformCharacteristics.....</b>	<b>15</b>
<b>18.TRC - WaveformDescriptions .....</b>	<b>18</b>
<b>19.TRC - PatternCharacteristics .....</b>	<b>20</b>
<b>20.TRC - NameChecks block.....</b>	<b>22</b>
<b>21.Waveform Generator Model.....</b>	<b>24</b>
<b>22.Fluid Concepts in Parameter Specification .....</b>	<b>28</b>
<b>Annex A: CTL Description of a distributed-resource Tester.....</b>	<b>31</b>
<b>Annex B: CTL Description of a structural tester .....</b>	<b>34</b>
<b>Annex C: Test Resourse Constraints Example - SC212.....</b>	<b>37</b>
<b>Annex D: Test Resourse Constraints Example - Agilent.....</b>	<b>39</b>
<b>Annex E: TestResourceConstraints Example .....</b>	<b>40</b>

## 1. Overview

Transferring “tester independent” test program/pattern represented in STIL to a specific ATE system is a desirable capability. This is required to be able to completely and unambiguously specify how the STIL program/patterns are mapped into the tester resources. For example usage of 2 bits for a signal to double the transfer rates vs usage of mux

for getting the same result.

Figure 1: STIL3 Usage Model (Tester targeting)

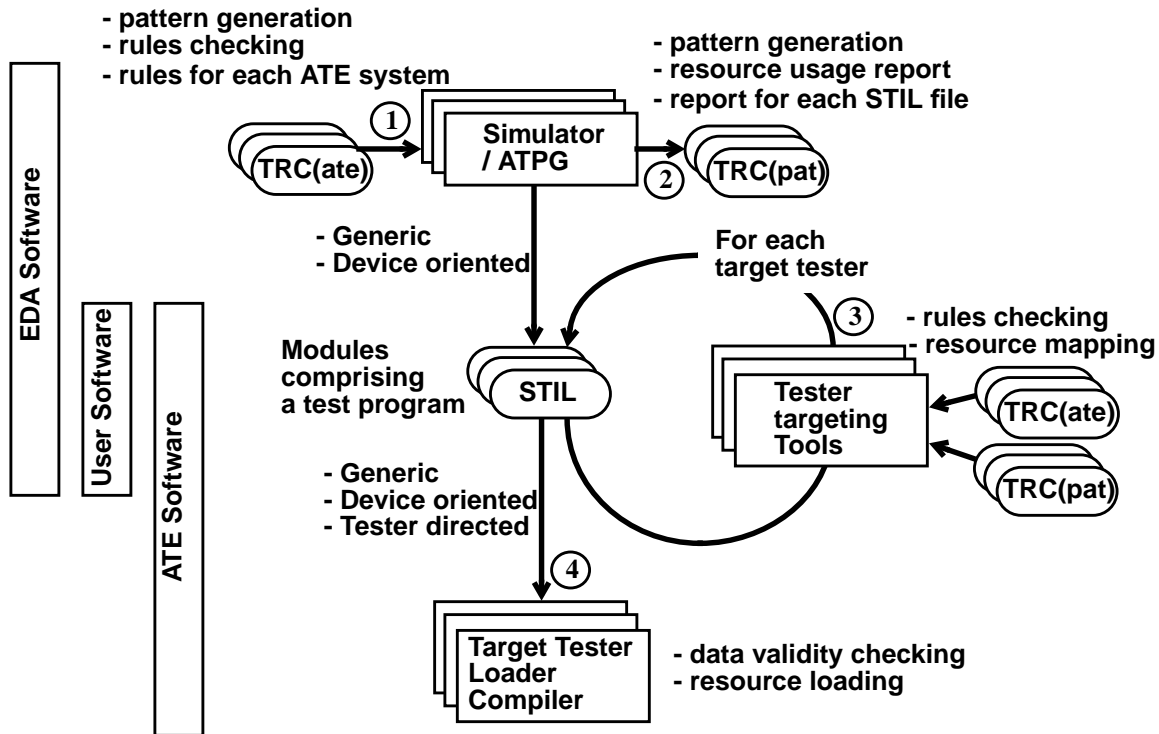


Figure 1 shows the usage model for Tester Targeting. The four ways that the TRC statements come into play in the flow of data from design to test are indicated by the circled numbers in the diagram. These four uses are as follows:

- (1) **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(s) is identified by means of the TRC file(s) defining the target tester(s).
- (2) **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: a) when a pattern is for an embedded core to be integrated into a chip, or b) for tester scheduling purposes. Each resource report is associated with a particular STIL file. The resource report data may be a separate file (as implied in the above diagram) or may be included in the STIL pattern file.
- (3) **Tester Resource Targeting** - The process of tester targeting is that of adding additional information into the STIL files that specifies how the resources of a given tester are to be assigned. Please note the bars on the left side of the diagram which indicate that this targeting operation could be done in one of

three places: a) by the EDA software that generates the patterns, b) by software created by the test user, or c) by the ATE software that loads the STIL patterns.

- (4) **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 an optimal job of assigning resources. If targeting information is present, then it is to be used to direct the resource assignment.

## 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 co-exist.
- 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.

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

- 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.

## 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. Due to 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 are to 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.

## 2. References

This standard shall be used in conjunction with the following standards. When the following standards are superseded by an approved version, the revision shall apply. These references are automatically updated during the editing process.

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

IEEE Std. 1450-1999, IEEE Standard Test Interface Language (STIL) for Digital Test Vectors.

IEEE Std. P1450.1, IEEE Standard for Design Interface Language (under development as of June 25, 2000).

IEEE Std. P1450.6, IEEE Standard for Core Test Language (under development as of December 22, 2000).

### 3. Definitions, acronyms, and abbreviations

#### 3.1 Definitions

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

**STIL0** Refers to IEEE Std. 1450-1999. This base STIL standard is commonly referred to as "dot 0" to differentiate it from all the extensions such as this 1450.3 extension).

**STIL1** Refers to IEEE Std. P1450.1. This base STIL standard is commonly referred to as "dot 1" to differentiate it from all the extensions such as this 1450.3 extension).

**STIL3** Refers to IEEE Std. P1450.3 (i.e., this standard). This base STIL standard is commonly referred to as "dot 3" to differentiate it from all the other extensions).

**STIL6** Refers to IEEE Std. P1450.6. This base STIL standard is commonly referred to as "dot 6" to differentiate it from all the extensions such as this 1450.3 extension).

### 4. Structure of this standard

This document is an adjunct to IEEE Std. 1450-1999. The conventions established and defined in IEEE Std. 1450-1999 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. 1450-1999 document and are so identified in the title. The constructs defined in this document are limited to the Environment block as defined by IEEE 1450.1. All clauses in this document are normative. Example code is provided within each clause. More complete examples are provided in the Annexes which are informative.

The following is a copy of the conventions as defined in IEEE Std. 1450-1999 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 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).

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

- In the Syntax definition sections of the document, each statement or group of statements is identified by a syntax line number (in paranthesis at the right side of the page). These number are to be referenced to the definitions that follow which contain the syntax line numbers in parenthesis on the left side of the page.
- In the Syntax definition section of the document, there are paragraphs preceeded by the indentifier “*TRC-ERR-#*”. These definitions are definitions of potential constraint errors that may occur when a pattern file is mapped to a specific set of TRC rules.
- 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.

## 5. TRC orientation and capabilities tutorial (informative)

*May 24, 2002, from Dan Fan, (Per-Pin Architecture)*

There are four usage models to use the TRC statements (refer to Figure 1) - (1) Tester Rule Checking, (2) Tester Resource Reporting, (3) Tester Resource Targeting, and (4) Tester Resource Loading. The (2) and (4) are typically resident in the corresponding STIL pattern while the usage model (1) and (3) are stand-alone files.

This clause illustrates with a simplified model of a per-pin architecture tester for the usage model (1) and (3). This pseudo tester has 4 clock channels with 2 drive edges per period. The clocks have fixed format of either RTZ or RTO. The vector period is up to 200MHz (5ns). This tester also has 160 data channels with single event per vector period. The regular vector memory has 16 million vectors and 2K subroutine memory.

A simplified STIL pattern to exercise a subset of behavior for an octal bus transceiver design, modeled after a TTL LS245. Details of this design can be found in the Annex E. of IEEE std 1450-1999 standard. This simplified example is matched to the tutorial of IEEE std 1450-1999 (Clause 5 of IEEE std 1450-1999).

### 5.1 Test Rule Checking for a Tester Model

The TRC description to specify this tester which consists of 4 clock channels and 160 regular data channels for the usage model (1) and (3). Assuming the clock channels are inputs only and have 16 Meg vectors behind each channel. The data channels can be either inputs or outputs and also have 16 Meg vectors behind each channel. This tester has single period generator with both resolution and accuracy of 10ps for the usage model (1) and (3). Assuming the period value can be programmed between 5ns to 4us.

```
1: STIL 1.0 {
```

```

2:   TRC 2002
3: }
4:
5: Spec StructuralTester_vars {
6:   Category StructuralTester_category {
7:     PeriodVal {Min '5ns', Max '4us'}
8:   } // end Category
9: } // end Spec
10:
11:TRC StructuralTester_rules {
12:  PeriodCharacteristics PeriodInfo {
13:    Category StructuralTester_category;
14:    Accuracy '10ps';
15:    MaxPeriods 1;
16:    MaxPeriodGenerators 1;
17:    PeriodTimeLimit PeriodVal;
18:    Resolution '10ps';
19:  } // end PeriodCharacteristics
20:
21:  WaveformCharacteristics ClockWav {
22:    Accuracy Edge '150ps';
23:    Accuracy EdgetoEdge '300ps';
24:    DriveEvents U D D/U 1;
25:    FormatSelect In {
26:      MaxShapes 2 Static; //Clock can be RTZ or RTO format
27:      MaxData Drive 2 Dynamic;
28:      MaxTimeSets 64 Dynamic { PerGroup 1024; }
29:    } // end FormatSelect
30:    MaxEvents Drive 1 ;
31:    Resolution '10ps';
32:  } // End WaveformCharacteristics ClockWav
33:
34:  WaveformCharacteristics DataWav {
35:    Accuracy Edge '500ps';
36:    Accuracy EdgetoEdge '1ns';
37:    CompareEvents Edge H/L/X 1;
38:    DriveEvents U/D/Z 1;
39:    FormatSelect InOut {
40:      MaxShapes 64 Dynamic;
41:      MaxData DriveCompare 3 Dynamic;
42:      MaxTimeSets 64 Dynamic { PerGroup 1024; }
43:    } // end Formats DataWav
44:    MaxEdgeTime '4 * PeriodVal';
45:    MaxEvents Drive 1;
46:    MaxEvents Compare 1;
47:    Resolution '40ps';
48:  } // End Data Waveform Characteristics
49:
50:  WaveformDescriptions ClockFormat Explicit {
51:    In RTZ_Format {
52:      Shape {
53:        D/U;
54:        RTZ_E2: D;
55:      } // end Shape

```

```

56:     } // end In RTZ_Format
57:     In RTO_Format {
58:         Shape {
59:             D/U;
60:             RTO_E2: U;
61:         } // end Shape
62:     } // end In RTO Format
63: } // end WaveformDescriptions
64:
65: Signals Clock {
66:     MaxSignals 4;
67:     DriveState U D;
68:     InOut Static;
69:     MaxVectorMemory '16*1024*1024';
70:     PeriodCharacteristics PeriodInfo Synch;
71:     WaveformCharacteristics ClockWav;
72:     WaveformDescriptions ClockFormat;
73: } // end Signals
74:
75: Signals Data In Out {
76:     MaxSignals 160;
77:     CompareStrobe Edge;
78:     CompareState H L X;
79:     DriveState U D Z;
80:     InOut WithinCycle;
81:     MaxVectorMemory '16*1024*1024';
82:     PeriodCharacteristics PeriodInfo Synch;
83:     WaveformCharacteristics DataWav;
84: } // end Signals
85: } // end TRC Rules

```

NOTE 5: through 9: - The **Spec** block is available in IEEE std 1450-1999 (as Clause 19). It is used to provide the corresponding period value range between 5ns to 4us.

NOTE 12: through 19: - The **PeriodCharacteristics** block contains all the property of the period generator of the corresponding ATE or ATE family. This specific ATE has single period generator with 10ps accuracy and resolution. The period value range is bounded within 5ns to 4us. It also calls out that the tester only has single period value. In other words, there is no period values switching on-the-fly capability.

NOTE 21: through 32: - The **WaveformCharacteristics ClockWav** block contains all the clock channel's characteristics of this tester. The clock channels have fixed formats - RTZ or RTO with 150ps edge accuracy. There are 64 timing sets available for each clock channel.

NOTE 34: through 48: - The **WaveformCharacteristics DataWav** block contains all the data channel's characteristics of this tester. The data channel only has single event per cycle with 500ps edge accuracy. There are 64 timing sets available for each channel.

NOTE 50: through 63: - The **WaveformDescriptions** block defines the clock channel's formats of this tester.

NOTE 65: through 73: - The **Signals Clock** block defines the corresponding property of the clock channels. In the usage model (1) Tester Rules Checking, it defines the clock channels of a particular tester's capability. In the usage model (3) Tester Resource Targeting, it defines the clock channels of a target tester (the configuration information). In this context, this tester has 4 clock channels available (**MaxSignals**). These channels can only have high (U) or low (D) state (**DriveState**) and can only support Input/Output state switch prior pattern execution (statically) - **InOut** statement. This particular tester has up to 16 Mega vectors behind each clock channel. The rest of statements refer to the period and waveform characteristics.

NOTE 75: through 84: - The **Signals Data In Out** block defines the corresponding property of the data signals. In the usage model (1) Tester Rules Checking, it defines the data channels of a particular tester's capability. In the usage model (3) Tester Resource Targeting, it defines the data channels of a target tester (the configuration information). This tester has 160 data channels available. The **CompareStrobe** statement defines this tester can only support edge strobe (i.e. no window strobe capability). These strobe events can have compare high (H), compare low (L) or don't compare (X) state (**CompareState**). the driver states can only have high (U), low (D), or don't drive (tri-state, Z) state (**DriveState**). This tester's data channel can support Input/Output state switch prior pattern execution (statically) - **InOut** statement. This tester has up to 16 Mega vectors behind each data channel.

## 5.2 Test Resource Usage in a STIL Pattern

The TRC statements that specify the pattern characteristics for this simplified TTL LS245 device will be located in the corresponding STIL pattern file. This simplified model has unidirectional bus - "A" bus signals are defined as inputs and "B" bus signals are defined as outputs. This example matches with the tutorial of IEEE std 1450-1999 Clause 5, refer to the Figure 3 of IEEE std 1450-1999 at page 9 for the STIL pattern. These TRC statements for the usage model (2) and (4) will represent as:

```

86:STIL 1.0 {
87:  TRC 2002
88:}
89:
90:TRC LS245_Resources {
91:  PeriodCharacteristics PeriodInfo {
92:    Accuracy '1ns';
93:    PeriodTimeLimit '500ns';
94:  } // end PeriodCharacteristics
95:
96:  WaveformCharacteristics RTOWav {
97:    Accuracy Edge '500ps';
98:    DriveEvents U D/U 1;
99:    FormatSelect In {
100:      MaxShapes 1 Static; //OE_ needs be RTO format
101:      MaxData Drive 2 Dynamic;
102:    } // end FormatSelect
103:  } // End WaveformCharacteristics RTOWav
104:
105:  WaveformCharacteristics NRZWav {
106:    DriveEvents U/D 1;
107:    FormatSelect In {
108:      MaxShapes 1 Static; //DIR and ABUS only have RTZ format
109:      MaxData Drive 2 Dynamic;
110:    } // end Formats DataWav
111:  } // End Data Waveform Characteristics
112:
113:  WaveformCharacteristics WindowWav {
114:    CompareEvents h/l/t 1;
115:    DriveEvents Z;
116:    FormatSelect Out {
117:      MaxData Compare 3 Dynamic;
118:    } // end Formats DataWav
119:  } // End Data Waveform Characteristics
120:
121:  WaveformDescriptions RTOFormat Explicit {
122:    In RTO_Format {

```

```

123:     Shape {
124:         D/U;
125:         U '@+100ns';
126:     } // end Shape
127: } // end In RTO Format
128: } // end WaveformDescriptions
129:
130: Signals Clock {
131:     DriveState U D;
132:     MaxVectorMemory 9;
133:     PeriodCharacteristics PeriodLS245 Synch;
134:     WaveformCharacteristics RTOWav;
135:     WaveformDescriptions RTOFormat;
136: } // end Signals
137:
138: Signals Data In {
139:     MaxSignals 9;
140:     DriveState U D;
141:     MaxVectorMemory 9;
142:     PeriodCharacteristics PeriodLS245 Synch;
143:     WaveformCharacteristics NRZWav;
144: } // end Signals
145:
146: Signals Data Out {
147:     MaxSignals 8;
148:     CompareStrobe Window;
149:     CompareState H L;
150:     MaxVectorMemory 9;
151:     PeriodCharacteristics PeriodLS245 Synch;
152:     WaveformCharacteristics WindowWav;
153: } // end Signals
154: } // end TRC Usage of LS245

```

NOTE 91: through 94: - The **PeriodCharacteristics** block contains all the requirements of period for the LS245's STIL pattern. The period value required by the DUT and this STIL pattern is 500 ns.

NOTE 96: through 103: - The **WaveformCharacteristics RTOWav** block contains all the signal OE\_'s characteristics of LS245. The OE\_ signal needs to have single RTO format with single timing values (200ns Down and 300ns Up).

NOTE 105: through 111: - The **WaveformCharacteristics NRZWav** block contains all the signal characteristics of DIR and ABUS of LS245. These signals only need have single event (or NRZ format) with single timing values (0ns or 10ns).

NOTE 113: through 119: - The **WaveformCharacteristics WindowWav** block contains all the signal characteristics of BBUS of LS245. The BBUS signal needs to have window strobe with single timing values (260ns - 280ns). It needs T state to turn off window strobe. It may needs to have the **DriveEvents** statement, if the signal BBUS is declared as an InOut signal (as page 14 of 1450-1999). The **DriveEvents** statement can be omitted, if the signal BBUS is declared as an Out signal (as page 9 of 1450-1999).

NOTE 121: through 128: - The **WaveformDescriptions** block defines the OE\_'s formats of LS245.

NOTE 130: through 136: - The **Signals Clock** block defines the corresponding property of the signal OE\_. The **DriveState** statement describes this signal only need ForceUp (U)/ForceDown (D) states. In this example, it has 9 vectors for this signal (**MaxVectorMemory** statement). The rest of statements refer to the period and waveform characteristics. The keyword Synch indicates all signals always synchronous together.

NOTE 138: through 144: - The **Signals Data In** block defines the corresponding property of the data input signals - DIRS and ABUS. The **MaxSignals** statement indicates the STIL pattern has 9 input signals which will be treated as data inputs, e.g. the DIR and ABUS signals. It refers to a predefined WaveformCharacteristics named as “NRZWav”.

NOTE 146: through 153: - The **Signals Data Out** block defines the corresponding property of the BBUS signals. The **MaxSignals** statement indicates the STIL pattern has 8 signals. The **CompareStrobe** statement defines the STIL pattern uses window strobe for the BBUS. The **CompareState** statement defines the STIL pattern only has compare high (H) and compare low (L) states. The **WaveformCharacteristics** statement refers to a predefined WaveformCharacteristics named as “WindowWav”.

## 5.3 The Process of Tester Targeting

In the process of tester targeting, these two pieces of information will help the user to port a tester-independent STIL pattern to a targeted-tester loadable STIL pattern.

### 5.3.1 The Period's Statements

All the hard resource requirements must be met, such as: MaxPeriods and PeriodTimeLimit. Some of soft requirements may be compromised to fit to a tester, such as Accuracy and Resolution.

The following conditions are well covered by the pseudo tester:

- (a) **Accuracy:** The tester has 10ps accuracy. The STIL pattern only needs 1ns accuracy.
- (b) **PeriodTimeLimit:** 500ns is within the boundary of 5ns to 4us range, so it is OK.
- (c) Others are not specified in STIL's PeriodCharacteristic block. These property will be treated as 'No Check' is needed.

### 5.3.2 The Signal's Statements

All the hard resource requirements must be met, such as: MaxSignals and MaxVectorMemory. Some of soft requirements may be compromised to fit to a tester, such as CompareStrobe.

The following conditions are well covered by the pseudo tester:

- (a) **MaxSignals:** The tester has 4 clock channels and 160 data channels. The STIL pattern only needs 1 clock and 17 data signals.
- (b) **MaxVectorMemory:** The tester has 16 Mega vectors behind each channels. The STIL pattern only needs 9 vectors.
- (c) **DriveState:** The tester can handle U/D/Z while STIL pattern only uses U/D.
- (d) **InOut:** The tester can handle dynamic I/O switching within test cycle. The STIL pattern has only unidirectional bus (no I/O switching is needed, so no InOut statement is used).
- (e) **CompareState:** The tester can handle H/L/X while STIL pattern only uses H/L.

The following condition is not fully satisfied by the pseudo tester:

- (a) **CompareStrobe:** The tester only can provide edge strobe, but the STIL pattern requests to have window strobe. The user may relax the STIL pattern to use edge strobe, then a new STIL pattern will be generated with edge strobing. The user may insist to have window strobe, then he/she will search for another tester.

If it is decide to compromise the STIL pattern to fit this particular tester, then the Signals Data Out block will be modified as the following:

```

155: WaveformCharacteristics EdgeWav {
156:   CompareEvents Edge H/L 1;
157:   FormatSelect Out {
158:     MaxData Compare 2 Dynamic;
159:   } // end Formats DataWav
160:   MaxEvents Compare 1;
161: } // End Data Waveform Characteristics
162:
163: Signals Data Out {
164:   MaxSignals 8;
165:   CompareStrobe Edge;
166:   CompareState H L;
167:   MaxVectorMemory 9;
168:   PeriodCharacteristics PeriodLS245 Synch;
169:   WaveformCharacteristics EdgeWav;
170: } // end Signals
    
```

## 6. STIL Syntax Description” - Extensions to STIL0 Clause 6

All constructs and restrictions for IEEE Std. 1450-1999 Clause 6 are in effect here, with the following additions:

*TBD: Are there any new constructs or restrictions? see STIL.1 for examples of ...*

### 6.1 Additional reserved words

*TBD: Add the CTL keywords to the following table.*

Table 1 lists all STIL reserved words defined by this standard and not defined in IEEE Std. 1450-1999. Subsequent clauses in this standard identify the use and context of each of these additional reserved words.

Table 2: Additions to STIL Reserved Words


### 6.2 Additional reserved characters

*TBD: Are there any new reserved characters?*

Several reserved characters identified in IEEE Std. 1450-1999 are applied in additional contexts for this standard. Table 2 lists additional STIL reserved characters defined in this standard as well as additional contexts of previously identified reserved characters Boldface text identifies extended applications defined in

this standard.

Table 3: Additions to STIL Reserved Characters

Char	Usage

## 7. Statement Structure and Organization

As shown in “STIL3 Usage Model (Tester targeting)” on page 3 there are four ways in which TRC blocks and statements can be utilized to accomplish various objectives. The table below 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 TRC block that is used contain a PatternReport is typically part of the STIL file 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 from the STIL file 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(s).

Table 4: STIL/TRC Block Usage

Block/Statement/Function	Purpose	C <sup>a</sup>	R <sup>b</sup>	T <sup>c</sup>	L <sup>d</sup>
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 { SystemCharacteristics { } } }	define system application	X			X
Environment { TRC { NameChecks { } } }	define name constraints	X			
Environment { TRC { PatternCharacteristics { } } }	define pattern information	X	X		X
Environment { TRC { PeriodCharacteristics { } } }	define period information	X	X		X
Environment { TRC { SignalCharacteristics { } } }	define signal information	X	X		X
Environment { TRC { WaveformCharacteristics { } } }	define waveform information	X	X		X
Environment { TRC { WaveformDescriptions { } } }	define wave shapes	X	X		X
STIL blocks	any block in STIL			X	X

Table 4: STIL/TRC Block Usage

Block/Statement/Function	Purpose	C <sup>a</sup>	R <sup>b</sup>	T <sup>c</sup>	L <sup>d</sup>
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

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

<sup>b</sup>Report = Information that is used to report the actual parameters of a given pattern or pattern burst.

<sup>c</sup>Tester Target = Information that is added to a STIL file to specify how it is to be loaded into the resources of a given ATE system.

<sup>d</sup>Tester Loading = Information that is used in a STIL file to control the loading of the resources of an ATE system.

## 7.1 TRC Usage for ATE Constraint Specification

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

```

2: STIL 1.0 ( TRC 2003; )
3: Environment ATE_1 {
4:   TRC {
5:     Usage Constraints;
6:   }
7: } // end Environment for ATE_1
8: Environment ATE_2 {
9:   TRC {
10:    Usage Constraints;
11:  }
12:} // end Environment for ATE_2

```

## 7.2 TRC Usage for Design/Pattern Constraints

A TRC file that is to be used for the constraining the creation of a design or pattern set for that design looks much the same as the ATE constraint file. The difference being that there will typically only be one set of constraints 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:

```

13:STIL 1.0 ( TRC 2003; )
14:Environment {
15:  TRC {
16:    Usage Constraints;
17:  }
18:} // end Environment

```

### 7.3 TRC Usage for Pattern Reporting

The same syntax as is used for specifying resource constraints on a pattern can also be used to report the resource utilization for that pattern (or pattern-burst). The main difference being that identification as such in the TRC statement. Some resource constraint blocks don't 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:

```

19:STIL 1.0 ( TRC 2003; )
20:Environment PATNAME_1 {
21:  TRC {
22:    Usage PatternReport;
23:  }
24:} //end Environment
25:Environment PATNAME_2 {
26:  TRC {
27:    Usage PatternReport;
28:  }
29:} //end Environment

```

### 7.4 TRC Usage for Tester Targetting

Tester targetting is accomplished by annotating a STIL file 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:

```

30:STIL 1.0 ( TRC 2003; )
31:Signals { sig[1..5] InOut; }
32:Environment ATE1 {
33:  NameMaps { } //specify signal to channel mapping
34:}
35:SignalGroups { sigs = 'sig[1..5]'; }
36:Timing basic {
37:  WaveformTable one {
38:    <perl> Period '500ns'; //tag the period resource
39:    DIR { <seq1> 01 { '0ns' D/U; }} //tag the per pin waveform resource
40:  }
41:  WaveformTable two {
42:    <perl> Period '500ns'; //use same resource as wft one
43:    DIR { <seq2> 01 { '0ns' D/U; }} //use different resource from wft one
44:  }
45:Pragma ATE1 {*
46:  map1: ...
47:  map2: ...
48:*}
49:Pattern P {
50:  V <map1> { sigs = 10101; } //select map1 from pragma ATE1
51:  V <map2> { sigs = LHLHL; } //select map2 from pragma ATE1
52:  V <map1> { sigs = 01010; } //select map1 from pragma ATE1
53:}

```

## 8. STIL statement - Extensions to STILO clause 8

The STIL statement identifies the primary version of IEEE Std. 1450-1999 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. 1450-1999.

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

All other constructs and restrictions for IEEE Std. 1450-1999 clause 8 are in effect here.

### 8.1 STIL syntax

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

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

IEEE\_1450\_0\_IDENTIFIER: The primary version of STIL, as identified by IEEE Std. 1450-1999.

(2) EXT\_NAME: The specific name of the Extension. This standard is identified by the name **TRC**.

EXT\_VERSION: The primary version of an EXT\_NAME. This standard is identified by the value **2002**.

### 8.2 STIL example

```
54:STIL 1.0 {
55:  Design 2002;
56:  TRC D04;
57:}
```

## 9. Resource Statement

The resource statement is used to identify on a vector basis specific resources within a target tester that are to be assigned for this vector. This information is optional and may be computed by the software load operation on the tester. See “STIL3 Usage Model (Tester targeting)” on page 3. 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.

### 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> labels 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> label. The Resource statement and the <RESOURCE\_ID> labels 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 is a label that may optionally be placed prior to any block or statement keyword. The purpose of this <RESOURCE\_ID> label is two fold: 1) it is used to identify the tester resource that is to be assigned to support the labelled block or statement, and 2) 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 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.

The format of the resource id within the angle bracket is not defined in this standard, but is to take on the format as required by the tester loader. Typically it will be: a) an integer, b) an alpha-numeric string containing 0..9, A..Z, a..z, \_, or c) a double quoted string that may contain special characters.

## 9.2 Example of Resource Assignment in a Pattern Block

```

1: STIL 1.0 { Design D14; TRC D04; }
2: Header {
3:   Source "IEEE P1450.3, Working-Draft 04, Aug 30, 2002";
4:   Ann {* clause 9.2 *}
5: }
6:
7: Signals { s[1..100] InOut; }
8:
9: // Load resource mapping memory for tester 1
10: Pragma tester1 {*
11:   L1: ... tester statement ...
12:   L2: ... tester statement ...
13:   L3 &x: ... tester statement ...
14: *} // end Pragma tester1
15:
16: // Load resource mapping memory for tester2
17: Pragma tester2 {*
18:   (0) ... tester statements ...
19:   (5) ... tester statements ...
20:   (13) ... tester statements ...
21: *} // end Pragma tester2
22:
23: Pattern P1 {
24:   Resource tester1 tester2;
25:       W wft1;
26:       C { s[1..100] = \r100 0; }
27:   <L1 0> V { s[5] = 1;
28:   >L2 13> V { s[1..4] = 1011; }
29:   <L1 5> V { s[10,11] = HH; }
30:   <"L3 &x" 13> V { s[1..4] = 0011; }
31: } // end Pattern P1

```

### 9.3 Example of Resource Assignment in a Timing Block

The following example shows the use of <resource\_id> labels within a timing block. See also “Example of Implicit Resource Assignment” on page 56 for an example of how to implicitly define common resources using the Inherit statement.

```

1: STIL 1.0 { Design D14; TRC D04; }
2: Header {
3:   Source "IEEE P1450.3, Working-Draft 04, Aug 30, 2002";
4:   Ann { * clause 9 * }
5: }
6:
7: Timing basic {
8:   WaveformTable one {
9:     <per1> Period `500ns`;
10:    DIR { <seq1> 01 { `0ns` D/U; }}
11:    OE_ { 01 { `0ns` U; `200ns` D/U; `300ns` U; }}
12:    ABUS { <seq1> 01 { `10ns` D/U; }}
13:    BBUS { <seq1> LHX { `0ns` Z; `260ns` L/H/X; `280ns` T; }}
14:  } // W one
15:  WaveformTable two {
16:    <per1> Period `500ns`;
17:    DIR { <seq2> 01 { `0ns` D/U; }}
18:    OE_ { 01 { `0ns` U; `200ns` D/U; `300ns` U; }}
19:    ABUS { <seq2> LHZX { `0ns` Z; `260ns` l/h/t/x; `280ns` x; }}
20:    BBUS { <seq2> 01 { `10ns` D/U; }}
21:  } // W two
22:  WaveformTable three {
23:    <per2> Period `550ns`;
24:    DIR { <seq1> 01 { `0ns` D/U; }}
25:    OE_ { 01 { `0ns` U; `200ns` D/U; `300ns` U; }}
26:    ABUS { <seq3> 01 { `10ns` D/U; }}
27:    BBUS { <seq3> LHX { `0ns` Z; `260ns` L/H/X; `280ns` T; }}
28:  } // W three
29:  WaveformTable four {
30:    <per_3> Period `550ns`;
31:    DIR { <seq2> 01 { `0ns` D/U; }}
32:    OE_ { 01 { `0ns` U; `200ns` D/U; `300ns` U; }}
33:    ABUS { <seq4> LHX { `0ns` Z; `460ns` L/H/X; `480ns` T; }}
34:    BBUS { <seq4> 01 { `10ns` D/U; }}
35:  } // W four
36:  WaveformTable five {
37:    <per1> Period `500ns`;
38:    DIR { <seq1> 01 { `0ns` D/U; }}
39:    OE_ { 01 { `0ns` U; `200ns` D/U; `300ns` U; }}
40:    ABUS { <seq2> LHX { `0ns` Z; `260ns` L/H/X; `280ns` T; }}
41:    BBUS { <seq1> LHX { `0ns` Z; `260ns` L/H/X; `280ns` T; }}
42:  } // W five
43:} // Timing basic
44:
45:Pattern basic {
46:  W five; V { ALL = 00ZZZZZZZZXXXXXXXX; }
47:  W one; V { ABUS = 00000000; BBUS = LLLLLLLLL; }
48:  W three;V { ABUS = 10000000; BBUS = LHLLLLLLL; }

```

```

49:  W three;V { ABUS = 00001000; BBUS = LLLLLLHLL; }
50:  W two;  V { DIR = 1; ABUS = LLLLLLHLL; BBUS = 00001000; }
51:  W two;  V { ABUS = LHLLLLLLL; BBUS = 10000000; }
52:  W four; V { ABUS = LHLLLLLLL; BBUS = 10000000; }
53:} // End Pattern basic

```

## 10. Pragma Block

A Pragma is a block of code that is implementation-dependant (i.e., it is code that is intended for use on a specific test system only). These blocks are a means of embedding tester specific instructions within a STIL file. As with standard annotations, the syntax within the pragma is not defined or limited in any way, except by the opening and closing brace/asterisk convention. The contents of the pragma is entirely up to the tester software that consumes it. This capability allows for a STIL generator that knows the target system(s) to put instructions in line with the STIL tester independent code, as opposed to creating side files for this purpose.

See the definition of the Resource statement in “Resource Statement” on page 16 for one way of using the Pragma block.

### 10.1 Syntax

```
Pragma TARGET_NAME { * ... ANY OLD SYNTAX ... * }
```

## 11. TesterChannelMap - Usage of STIL1, Clause 18

(informative)

In STIL.1, Clause 18, there is the definition of how to specify a mapping of Signals and other objects from the STIL pattern interchange environment to some other environment. This facility can be used to specify the mapping of a STIL test program to a set of tester channels on a tester. This clause is informational only as no new syntax is defined herein.

### 11.1 TesterChannelMap - Syntax

```
Environment (ENV_NAME) {
  ( NameMaps (MAP_NAME) {
    ( Signals { (SIG_NAME "MAP_STRING"); } ) *
  } // end NameMaps
} // end Environment

```

### 11.2 TesterChannelMap - Example

```

54:STIL 1.0 { Design D14; TRC D04; }
55:Header {
56:  Source "IEEE P1450.3, Working-Draft 04, Aug 30, 2002";
57:  Ann { * clause 11 * }
58:}
59:

```

```

60:Environment ATE_SC212 {
61:  NameMaps Wafer {
62:    Signals {
63:      // sig-name "channel, type";
64:      SIG1 "13, INPUT";
65:      SIG2 "45, BIDI";
66:      SIG3 "46, OUT";
67:    } // end Signals
68:  } // end NameMaps Wafer
69:  NameMaps Package {
70:    Signals {
71:      SIG1 "42, INPUT";
72:      SIG2 "19, BIDI";
73:      SIG3 "16, OUT";
74:    } // end Signals
75:  } // end NameMaps Package
76:  NameMaps MultiSite {
77:    Signals {
78:      SIG1 "INPUT 13 48 96";
79:      SIG2 "BIDI 45 83 99";
80:      SIG3 " OUT 46 85 128";
81:    } // end Signals
82:  } // end NameMaps MultiSite
83:  NameMaps MultiSite_pingpong {
84:    Signals {
85:      SIG1 "INPUT (13 48) (96 106)";
86:      SIG2 "BIDI (45 83) (99 107)";
87:      SIG3 "OUT (46 85) (128 126)";
88:    } // end Signals
89:  } // end NameMaps MultiSite_pingpong
90:} // end Environment

```

## 12. TRC - TestResourceConstraints Block

The TestResourceConstraint block is used to define characteristics of a tester or test environment. Multiple blocks may exist, each defining a separate tester or test environment, in which case it is the responsibility of the tool environment to choose which constraint(s) shall be applied.

Some blocks defined herein may be referenced by statements within other STIL standards documents (i.e., CTL to convey statistical information about a pattern. CTL -> PatternInformation -> PatternCharacteristics).

### 12.1 TRC Syntax

The following are general rules of interpretation that apply to all of the statements in this block:

- (a) When a statement is omitted, then this means that no checking of this constraint shall be done. Note: optional statements are contained within paranthesis ( ).
- (b) An integer value of '-1' means that for all intent and purpose the parameter is unbounded and therefore need not be checked.
- (c) When a statement has multiple parameters, they are treated as an 'AND' function (i.e., all conditions shall be met). For example: MaxEvents Drive 4 Compare 2;

- (d) When parameters are specified on separate statements, they are treated as an ‘OR’ condition (i.e., any of the conditions may be met). For example: MaxEvents Drive 4; MaxEvents Compare 2;

*TBD: The italic descriptions that follow the syntax definition are to be moved to the Syntax Description section and are to be elaborated. This is to be done after the syntax stabilizes.*

```

Environment { (1)
  TRC TRC_NAME { (2)
    // This block specifies the constraints of a given test environment.
    // The name is typically the name/model identifying a tester configuration.
    (Category (CAT_NAME)+ ;) (3)
    // Specify a list of categories containing variables used by this test constraints block.
    (NameChecks (NAM_CHK_NAME) { })* (4)
    (PatternCharacteristics PAT_CHAR_NAME { })* (5)
    (PeriodCharacteristics PER_CHAR_NAME { })* (6)
    (SignalCharacteristics (signal_attributes)+ { })* (7)
    (SystemCharacteristics { (8)
      (MultipleSites integer_expr;)
      (MultiplePorts integer_expr;)
    })
    (Usage < Constraints | PatternReport > ;) (9)
    (WaveformCharacteristics WAV_CHAR_NAME { })* (10)
    (WaveformDescriptions WAV_DESC_NAME (Explicit) { })* (11)
  } // end TestResourceConstraints
} // end Environment

```

(1) **Environment**: The Environment block is a general purpose construct defined in STIL.1 for the purpose of describing application environments for STIL data. In this case, the Environment block is used to describe tester resource constraint data that is useful for transferring data from an EDA (pattern generation) environment into an ATE (pattern consumption environment).

(2) **TRC** TRC\_NAME: This statement begins a block describing a set of constraints for a given tester application. The TRC\_NAME is typically the name/model of the ATE system that is being defined.

(3) **Category** (CAT\_NAME)+ : This statement is optional, and is used to define the category name(s) that contains floating point variables that are used within the TRC block. Note that in the Variables block (as defined in STIL.1) it is also possible to define integer variables and integer constants. The typical usage for variables within TRC is for “fluid” constraints, i.e., constraints that are coupled one to another.

(4) **NameChecks** (NAM\_CHK\_NAME): The NameChecks block is optional and contains statements describing naming rules for the objects within a STIL file. Refer to clause 53 for details.

(5) **PatternCharacteristics** PAT\_CHAR\_NAME: The PatternCharacteristics block is optional and contains information relative to the pattern objects in a STIL file. Refer to clause 18 for details.

(6) **PeriodCharacteristics** PER\_CHAR\_NAME: The PeriodCharacteristics block is optional and contains information relative to the period capabilities that can be specified in the Timing of a STIL file. Refer to clause 15 for details.

(7) **SignalCharacteristics** (signal\_attributes)+: The SignalCharacteristics block is optional and contains information relative to the signals (i.e., ATE pin channels) of a STIL file. Refer to clause 13 for details.

(8) **SystemCharacteristics**: The SystemCharacteristics block is used to contain statement with regard to the overall ATE system.

**MultipleSites** *integer\_expr*: Specify the number of identical, but independant copies of the program that can run simultaneously. i.e., the the number of identical devices that can be tested in parallel.

**MultiplePorts** *integer\_expr*: Specify the number of non-identical partitions that can be run simultaneously. i.e., the number of separate time domain tests that can be run in parallel. This may be in support of a device with separatly timed signals, or in support of independantly tested cores or devices.

(9) **Usage**: This statement is used to indicate the intent of the TRC block. The allowed identifiers are:

**Constraints**: Specify that the TRC block is used to define constraints that are to be used either to identify limitations of an ATE system or a set of constraints that are to be imposed on a design.

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

(10) **WaveformCharacteristics** *WAV\_CHAR\_NAME*: The WaveformCharacteristics block is optional and contains information about the waveforms that a particular ATE system can support. Refer to clause 16 for details. Please note that whereas this block describes characteristics or attributes of the waveforms, the WavefromDescriptions block describes actual waveforms that can be represented on an ATE system. Either method of describing waveforms may be used as appropriate to a given ATE system.

(11) **WaveformDescriptions** *WAV\_DESC\_NAME*: The WaveformDescriptions block is optional and contains a defined set of waveforms that a particular ATE system can support. Refer to clause 17 for details. Please note that whereas this block describes actual waveforms that can be represented on an ATE system, the WavefromCharacteristics block describes characteristics or capabilities of the waveforms. Either method of describing waveforms may be used as appropriate to a given ATE system.

## 13. TRC - SignalCharacteristics

### 13.1 TRC - SignalCharacteristics - Syntax

```

SignalCharacteristics (< Data | Clock | Scan | In | Out | SplitIO | Asynchronous | User
USER_DEFINED >)+ {
    ( FanOut integer_expr;)
    ( InOut < WithinCycle | OnCycleBoundary | Static >;)
    ( MaxScanMemory integer_expr;)
    ( MaxCaptureMemory integer_expr (<FailOnly | Result>)+ ;)
    ( MaxScanChainLength integer_expr ;)
    ( MaxSignals integer_expr;)
    ( MaxVectorMemory integer_expr;)
    ( PeriodCharacteristics PER_CHAR_NAME (<Synchronous | Asynchronous>)+ ;)
    ( WaveformCharacteristics WAV_CHAR_NAME ;)
    ( WaveformDescriptions WAV_DESC_NAME ;)
} // end SignalCharacteristics

```

(1) **SignalCharacteristics**: The SignalCharacteristics block defines the characteristics of a set of signals that have like functionality. There maybe multiple SignalCharacteristics blocks if there are different type of tester channels (pins) on an ATE system. Immediately after the SignalCharacteristics keyword there is a list of one or more signal type keywords that shall be selected from the following list:

- (a) **Data**: A data type of signal is for defining functional data. Data signals will typically have data provided from the pattern vectors, but will typically not have high timing accuracy requirements as would be provided by a Clock signal type. The additional type keywords <In|Out> can be added to define direction.

- (b) **Clock**: A clock type of signal is for defining clock data. Clock signals will typically have data provided from the pattern vectors, and will typically have high timing accuracy requirements. The additional type keywords <In|Out> can be added to define direction.
  - (c) **Scan**: A scan type of signal is used for defining scan (in/out) data.
  - (d) **In**: An input signal is capable of driving data into the device. For a bi-directional signal, include also the Out keyword.
  - (e) **Out**: An output signal is capable of comparing the output of the device. For a bi-directional signal, include also the In keyword.
  - (f) **SplitIO**: A split I/O signal is capable of driving and comparing on separate tester pin channels.
  - (g) **Asynchronous**: An asynchronous signal is one that is not synchronous to the period clock.
  - (h) **User** `USER_DEFINED`: A user defined signal type is used to cover any special cases not covered by the above types. Use of this capability is limited to tools that can support the given data type (in much the same way as UserKeyword and UserFunctions).
- (2) **FanOut** *integer\_expr*: The FanOut statement is used to define the number of tester pin channels that may be driven by a given signal.
- (3) **InOut**: The InOut statement is used to specify the in/out switching capabilities of a tester channel. Note that whereas the In/Out specification on the top level Signals block specifies the basic ability to either drive or compare, this statement specifies when the in/out switching may occur. The following are allowed:
- (a) **WithinCycle**: This keyword specifies that in/out switching may occur within a cycle (period). The exact time of the switch is specified by the WaveformTable and the drive/compare event times within the waveform definition.
  - (b) **OnCycleBoundary**: This keyword specifies that in/out switching may occur only on period boundaries. This effectively means that each waveform definition may have only drive or compare events, and that the first one must occur at T0 of the cycle.
  - (c) **Static**: This keyword specifies that the in/out cannot be switched during the execution of a pattern. For a static signal, it is typically established at the beginning of the pattern exec. See also the pattern statement "Fixed" as defined in STIL.1.
- (4) **MaxScanMemory** *integer\_expr*: This statement is used to specify that a scan state memory is available for this signal type and the number of scan states that can be defined for the signal. If this statement is omitted, it may still be possible to load scan states using vector memory.
- (5) **MaxCaptureMemory** *integer\_expr*: This statement is used to specify that there is a capture memory associated with the signal and the size of the capture memory. An additional keyword is used to specify how the memory is to be used. If both keywords are present, then the tester is capable of both.
- (a) **FailOnly**: The capture memory can be used to capture fail data.
  - (b) **Result**: The capture memory can be used to capture result data.
- (6) **MaxScanChainLength**: This statement is used to specify the maximum length of each scan chain.
- (7) **MaxSignals** *integer\_expr*: This statement is used to specify the maximum number of signals of a given type that are available on an ATE system.
- (8) **MaxVectorMemory** *integer\_expr*: This statement is used to specify the maximum memory available for the storage of vector state data for the given type of signal.
- (9) **PeriodCharacteristics** `PER_CHAR_NAME`: This statement is used to reference a named block that defines the PeriodCharacteristics for this type of signal. An additional keyword indicates whether to share period characteristics with other signal types. If both keywords are present then the tester is capable of both.
- (a) **Synch**: Signals of this type shall be synchronous with all other signals blocks that specify the same period characteristics block.
  - (b) **Asynch**: Signals of this type shall be asynchronous with all other signals blocks. i.e., these signals shall use a private reference period generator.

(10) **WaveformCharacteristics** WAV\_CHAR\_NAME: This statement is used to reference a waveform characteristics block (within the current TRC block) that defines the waveform generation capabilities of this signal type. Note that the waveforms may also be defined as shapes (via the WaveformDescriptions block) as well as by their characteristics. If both blocks exist, then the rules of both blocks shall be satisfied.

(11) **WaveformDescriptions** WAV\_DESC\_NAME: This statement is used to reference a waveform descriptions block (within the current TRC block) that defines the waveshapes that can be created by this signal type. Note that the waveforms may also be defined as characteristics (via the WaveformCharacteristics block) as well as by their waveforms. If both blocks exist, then the rules of both blocks shall be satisfied

## 13.2 TRC - SignalCharacteristics - Syntax Example

## 14. TRC - SignalCharacteristics Supply

*TBD: review 1450.2 for supply resource defs - AI Tony & Rohit*

```
SignalCharacteristics Supply {
} // end SignalCharacteristics Supply
```

## 15. TRC - PeriodCharacteristics

### 15.1 TRC - PeriodCharacteristics - Syntax

```
PeriodCharacteristics PER_CHAR_NAME { (1)
  (Accuracy time_expr;) (2)
  (MaxPeriods integer_expr;) (3)
  (MaxPeriodGenerators integer_expr (< Dynamic | Static >);) (4)
  (MaxPeriodGenerators integer_expr (< Dynamic | Static >)) {
    ( PerGroup integer_exp; )
  } // end MaxPeriodGenerators
  ( PeriodSelectMemory integer_expr; ) (5)
  ( PeriodSelectMemory integer_expr {
    ( PerGroup integer_exp; )
  } ) // end PeriodSelectMemory
  (PeriodTimeLimit time_expr (time_expr);) (6)
  (Resolution time_expr;) (7)
} // end PeriodCharacteristics
```

(1) **PeriodCharacteristics** PER\_CHAR\_NAME: The period characteristics block contains statements defining the attributes of the period (or cycle) generator of an ATE system. The name (PER\_CHAR\_NAME) of this block is used by a SignalCharacteristics block to reference the capabilities that apply to the signals.

(2) **Accuracy** time\_expr: Specify the accuracy of the ATE system in the generating the period. The accuracy means that the actual period shall be within (period - time\_expr < t > period + time\_expr). (Note that there are multiple factors that must be considered in the overall timing: a) period accuracy and resolution, b) drive event accuracy and resolution, and c) compare event accuracy and resolution. This statement specifies only the accuracy of the period.)

(3) **MaxPeriods** *integer\_expr*: This statement allows to specify the maximum number of period values that may be specified. The periods are to be selected on a vector basis by means of the associated waveform table.

(4) **MaxPeriodGenerators** *integer\_expr*: This statement allows to specify the number of independent periods that may be specified simultaneously. These independent period generators are to be assigned to blocks of signals that will then execute independantly according to the separate period generation sequence of each one. The following additional parameters shall be specified:

- (a) **Dynamic**: This keyword specifies that the signal assignment may be changed from one pattern exec to the next.
- (b) **Static**: This keyword specifies that the signal assignment to a period generator is fixed according to the architecture of the ATE system and cannot be changed.

The MaxPeriodGenerator may optionally be defined as a block and contain the following statement:

**PerGroup** *integer\_exp*: This statement specifies the number of signals that are associated with a period generator. Note that “PerGroup 1;” means that there is separate period generation for each signal.

(5) **PeriodSelectMemory** *integer\_expr*: This statement specifies that the period selection is accomplished by means of an indirect selection memory. The integer value species the size of the indirect memory. An optional statement may be specified:

**PerGroup** *integer\_exp*: This statement specifies the number of signals that are associated with a period select memory. Note that “PerGroup 1;” means that there is separate period selection for each signal.

(6) **PeriodTimeLimit** *time\_expr (time\_expr)*: This statement species the limits of the period generator. The first time expr is the minimum allowed time, and the optional second value is the maximum allowed time.

(7) **Resolution** *time\_expr*: This statement is used to specify the minimum increments that the period can be set to. Note that this is different from accuracy which defines the relation between the value programmed and the resulting effect on an ATE system.

## 15.2 TRC - PeriodCharacteristics - Example

# 16. TRC - WaveformCharacteristics

## 16.1 TRC - WaveformCharacteristics - Syntax

```

WaveformCharacteristics WAV_CHAR_NAME { (1)
  ( Accuracy <Edge|EdgeToEdge> time_expr;)* (2)
  ( CompareEvents (3)
    (<H|L|X|V|T|h|l|x|v|t>/(<H|L|X|V|T|h|l|x|v|t>)*+ (integer_expr) ;)
  ( DriveEvents (4)
    (<U|D|Z|P>/(<U|D|Z|P>)*+ (integer_expr) ;)
  ( FormatSelect < In | Out | InOut > { (5)
    ( MaxShapes integer_expr (< Dynamic | Static >); ) (6)
    ( MaxShapes integer_expr (< Dynamic | Static > ) {
      ( PerGroup integer_exp; )
    } ) // end FormatSelect
    ( MaxTimeSets integer_expr (< Dynamic | Static > ) ; ) (7)
    ( MaxTimeSets integer_expr (< Dynamic | Static > ) {
      ( PerGroup integer_exp; )
      ( PerTimingGenerator; )
    } ) // end MaxTimeSets

```

```

    ( MaxTimingGenerators integer_expr (< Dynamic | Static > ); )           (8)
    ( MaxTimingGenerators integer_expr (< Dynamic | Static > ) {
      ( PerGroup integer_exp; )
    } // MaxTimingGenerators
    ( MaxData <Drive|Compare|DriveCompare> integer_expr (<Dynamic|Static>); )   (9)
    ( MaxData <Drive|Compare|DriveCompare> integer_expr (<Dynamic|Static> {
      ( PerGroup integer_exp; )
    } // end MaxData
    ( MaxIO integer_expr integer_expr (< Dynamic | Static >); )           (10)
    ( MaxIO integer_expr integer_expr (< Dynamic | Static > ) {
      ( PerGroup integer_exp; )
    } // end MaxIO
    ( MaxMask integer_expr (< Dynamic | Static >); )                       (11)
    ( MaxMask integer_expr (< Dynamic | Static > ) {
      ( PerGroup integer_exp; )
    } // end MaxMask
  }* // end FormatSelect
  ( MaxEdgeTime time_expr; )                                               (12)
  ( MaxEvents (< Drive integer_expr | OnOff integer_expr | Compare integer_expr >)* ;)* (13)
  ( MinCompareWindow time_expr; )                                         (14)
  ( MinEdgeReTrigger time_expr (<Drive | Compare>)* ;)*                   (15)
  ( MinDriveOffTime time_expr; )                                           (16)
  ( MinDriveOnTime time_expr; )                                            (17)
  ( MinDrivePulse time_expr; )                                             (18)
  ( ReplicateSubWaveform integer_expr; )                                    (19)
  ( Resolution time_expr; )                                                (20)
  ( WaveformSelectMemory integer_expr ; )                                   (21)
  ( WaveformSelectMemory integer_expr {
    ( PerGroup integer_exp; )
    ( SelectWithPeriod; )
  } // end WaveformSelectMemory
} // end WaveformCharacteristics

```

(1) **WaveformCharacteristics** WAV\_CHAR\_NAME: This block describes key characteristics or attributes about the waveforms. The WAV\_CHAR\_NAME defines a name for this block that is used within a SignalCharacteristics block to reference the appropriate waveform characteristics block. Note: See the SignalCharacteristics block for the relationship between the WaveformCharacteristics block and the WaveformDescriptions block

(2) **Accuracy**: This statement is used to specify the accuracy of each edge (i.e., each event in a waveform).

- Edge**: This keyword specifies that the accuracy is relative to the beginning of the period.
- EdgeToEdge**: This keyword specifies that the accuracy is relative to any other timing event.
- time\_expr*: This is the value to be assigned to the accuracy. The occurrence of the event shall be within  $(\text{edge\_time} - \text{accuracy\_time}) < t < (\text{edge\_time} + \text{accuracy\_time})$ .

(3) **CompareEvents**: This statement defines all compare events and compare event sequences that shall be allowed within a waveform. If the list contains only upper case event identifiers, then only edge strobe is allowed. Likewise, if only lower case then only window strobe is allowed. If both types of events, then either edge or window is allowed.

- $\langle \text{H|L|X|V|T|h|l|x|v|t} \rangle$ : A list of the allowed single events shall be defined (i.e., H L X h l)
- $\langle \text{/H|L|X|V|T|h|l|x|v|t} \rangle$ : A list of allowed compound compare events shall be defined (i.e., H/L, H/L/T)
- integer\_expr*: This value defines the number of compound compare events that shall exist in a given waveform. The default is one. For example, if this integer is a '2' then the following waveform is

allowed: AB{‘10ns’ L/H; ‘15ns’ X; ‘20ns’ L/H; ‘25ns’ X;}, and two wfc states are expected from the pattern to specify each waveform instance.

(4) **DriveEvents:** This statement defines all drive events and drive event sequences that shall be allowed within a waveform.

- (a) **<U|D|Z|P>:** A list of allowed drive events shall be defined (e.g., U D Z)
- (b) **/<U|D|Z|P>:** A list of allowed compound drive events shall be defined (e.g., D/U).
- (c) **integer\_expr:** This value defines the number of compound drive events that shall exist in a given waveform. The default is one. For example, if this integer is a ‘2’ then the following waveform is allowed: pP{‘0ns’ P; ‘10ns’ D/U; ‘15ns’ D; ‘20ns’ D/U; ‘30ns’ Z;}, and two wfc states are expected from the pattern to specify each waveform instance.

(5) **FormatSelect:** This statement begins a block that defines the format selection attributes of a waveform. The total number of possible waveforms is the product of all the integers within this block. If select operations are combines (e.g., timing and I/O select is a common select operation on a given ATE system), then use the NumShapes statement only. The In|Out|InOut keyword in this statement shall coordinate with the keyword on a referencing SignalCharacteristic block (i.e., if a SignalCharacteristics block is of type “In”, then only FormatSelect of “In” shall be allowed).

- (a) **In:** This keyword specifies that this block contains waveforms with drive events only.
- (b) **Out:** This keyword specifies that this block contains waveforms with compare events only.
- (c) **InOut:** This keyword specifies that this block contains waveforms with both drive and compare events.

(6) **MaxShapes integer\_expr:** This statement specifies the total number of shapes that shall be allowed. If this statement is used, then it shall be the only one within this block. If this statement is omitted, then the other statement shall be used to specify the individual selections criteria.

- (a) **Dynamic:** Specify that shapes are selectable on a vector by vector basis
- (b) **Static:** Specify that shapes are selectable only at the beginning of a pattern exec.
- (c) **PerGroup integer\_exp:** Specify the number of signals that shall select shapes in common. If this statement is omitted, then selection is per-signal.

(7) **MaxTimeSets integer\_expr:** This statement specifies the number of time sets that shall be allowed. This determines the number of timing values that may be assigned to a given waveform in order to create formats that are of the same shape, but different timing.

- (a) **Dynamic:** Specify that time sets are selectable on a vector by vector basis.
- (b) **Static:** Specify that time sets are selectable only at the beginning of a pattern exec.
- (c) **PerGroup integer\_exp:** Specify the number of signals that shall select time sets in common. If this statement is omitted, then selection is per-signal.
- (d) **PerTimingGenerator:** Specify that time set selection is done on a per-TG basis. See MaxTimingGenerators statement for the further definition.

(8) **MaxTimingGenerators integer\_expr:** This statement specifies the number of timing generators that shall be allowed. A timing generator is a function that can generate multiple sets of timing events (i.e., time sets) for multiple signals.

- (a) **Dynamic:** Specify that TGs are selectable on a vector by vector basis.
- (b) **Static:** Specify that TGs are selectable only at the beginning of a pattern exec.
- (c) **PerGroup integer\_exp:** Specify the number of signals that shall select TGs in common. If this statement is omitted, then selection is per-signal.

(9) **MaxData:** This statement specifies the number of data values that are used to make up a waveform. This is typically referred to as pattern memory data. For example if the pattern memory has one bit per pin for both drive and compare, then this condition is defined by “DriveCompare 2;”, whereas if the pattern memory has separate data bits for drive and compare then it would be defined by “Drive 2; Compare 2;”.

- (a) **Drive integer\_expr:** Specify the number of data states available for drive.
- (b) **Compare integer\_expr:** Specify the number of data states available for compare.

- (c) **DriveCompare** *integer\_expr*: Specify the number of data states to be shared by drive and compare.
  - (d) **Dynamic**: Specify that data states are selectable on a vector by vector basis.
  - (e) **Static**: Specify that data states are selectable only at the beginning of a pattern exec.
  - (f) **PerGroup**: Specify the number of signals that shall select data states in common. If this statement is omitted, then selection is per-signal.
- (10) **MaxIO** *integer\_expr integer\_expr*: This statement specifies the number of input/output select values that are used to make up a waveform.
- (a) **Dynamic**: Specify that input/output selection is on a vector by vector basis.
  - (b) **Static**: Specify that input/output selection is only at the beginning of a pattern exec.
  - (c) **PerGroup**: Specify the number of signals that shall select input/output states in common. If this statement is omitted, then selection is per-signal.
- (11) **MaxMask** *integer\_expr*: This statement specifies the number of compare mask select values that are used to make up a waveform.
- (a) **Dynamic**: Specify that compare mask selection is on a vector by vector basis.
  - (b) **Static**: Specify that compare mask selection is only at the beginning of a pattern exec.
  - (c) **PerGroup**: Specify the number of signals that shall select compare mask states in common. If this statement is omitted, then selection is per-signal.
- (12) **MaxEdgeTime** *time\_expr*: Specify the maximum allowed time that an edge can be programmed from T0 (i.e., the beginning of the period).
- (13) **MaxEvents**: This statement specifies the maximum number of events that may used to make up a waveform.
- (a) **Drive** *integer\_expr*: Specify the maximum number of drive events allowed in a waveform (i.e., U, D).
  - (b) **OnOff** *integer\_expr*: Specify the maximum number of on/off events allowed in a waveform (i.e., Z, P)
  - (c) **Compare** *integer\_expr*: Specify the maximum number of compare events allowed in a waveform (i.e., L, H, T, V, X, l, h, t, v, x)
- (14) **MinCompareWindow** *time\_expr*: Specify the minimum allowed strobe width (i.e., from L or H to X). Note that the next occurrence may be in a following period.
- (15) **MinEdgeReTrigger** *time\_expr*: Specify the minimum time that shall exist prior to the next occurrence of a like event. Note that the next occurrence may be in a following period.
- (a) **Drive**: Specify that the retrigger time applies to drive events.
  - (b) **Compare**: Specify that the retrigger time applies to compare events.
- (16) **MinDriveOffTime** *time\_expr*: Specify the minimum allowed drive off time (i.e., from Z to next U, D, or P). Note that the next occurrence may be in a following period.
- (17) **MinDriveOnTime** *time\_expr*: Specify the minimum allowed drive on time (i.e., from U, D, or P to Z). Note that the next occurrence may be in a following period.
- (18) **MinDrivePulse** *time\_expr*: Specify the minimul allowe drive pulse width (i.e., from U to D, or D to U). Note that the next occurrence may be in a following period.
- (19) **ReplicateSubWaveform** *integer\_expr*: This statement specifies the number of times that the waveform may be repeated. The overall period is split into equal parts.

*Question: To Frances Miller - What is the purpose/application of the above stmt? mux mode? rep-clock? How does it tie in with other statements? Is it more for pattern reporting? or as a tester constraint?*

(20) **Resolution** *time\_expr*: This statement defines the minimum increment for specifying a time value of a waveform event. Note that there is a separate statement for specifying the period resolution.

(21) **WaveformSelectMemory** *integer\_expr*: This statement defines the size of an indirect memory that has the function of accessing the specific waveforms. If this statement is not present, then no indirect memory is present. The optional statement “**PerGroup** *integer\_exp*;” allows to specify the number of signals that share the indirect memory in common. If the PerGroups statement is not present, then the indirect memory is on a per-pin basis.

(22) **SelectWithPeriod**: This statement defines that the indirect memory specified by the WaveformSelectMemory statement is to be accessed in common with the indirect memory specified by the PeriodSelectMemory.

## 16.2 TRC - WaveformCharacteristics - Example

# 17. TRC - WaveformDescriptions

## 17.1 TRC - WaveformDescriptions - Syntax

```

WaveformDescriptions WAV_DESC_NAME (Explicit) { (1)
  (< In | Out | InOut > WFNAME { (2)
    (NumberData integer_expr;) (3)
    (NumberIO integer_expr;) (4)
    (NumberMask integer_expr;) (5)
    (NumberPeriods integer_expr;) (6)
    (NumberShapes integer_expr;) (7)
    (NumberSignals integer_expr;) (8)
    (NumberTimeSets integer_expr;) (9)
    Shape { (10)
      ((TRC_LABEL:) < EVENT | EVENT_LIST> ;)*
      ((TRC_LABEL:) < EVENT | EVENT_LIST> {
        (Min time_expr;) *
        (Max time_expr;) *
      })*
    } // end Shape
  })* // end < In | Out | InOut >
} // end WaveformDescriptions

```

(1) **WaveformDescriptions** WAV\_DESC\_NAME: This block contains descriptions of allowed waveform shapes and times. It also defines the waveform select resources consumed by each waveform. The optional **Explicit** parameter means only defined waveforms are allowed. This block is referenced by a SignalCharacteristic block by means of the WAV\_DESC\_NAME identifier. Note: See the SignalCharacteristics block for the relationship between the WaveformCharacteristics block and the WaveformDescriptions block.

(2) < **In** | **Out** | **InOut** > WFNAME: This statement begins a block that defines a waveform and the resources needed to create the waveform. A block beginning with the keyword In defines an input only waveform. A block beginning with the keyword Out defines an output only waveform. A block beginning with the keyword InOut defines a bi-directional waveform. The WFNAME is for informational purpose only and is to apply a descriptive name to the waveform.

- (3) **NumberData** *integer\_expr*: Specify the number data values used to create this waveform. Default = no check.
- (4) **NumberIO** *integer\_expr*: Specify the number IO controls used to create this waveform. Default = no check.
- (5) **NumberMask** *integer\_expr*: Specify the number mask controls used to create this waveform. Default = no check.
- (6) **NumberPeriods** *integer\_expr*: Specify number periods needed to create this shape. Default = no check.
- (7) **NumberShapes** *integer\_expr*: Specify the number shape selects used to create this waveform. Default = no check.
- (8) **NumberSignals** *integer\_expr*: Specify the number of tester pins needed to create this shape. Default = no check.
- (9) **NumberTimeSets** *integer\_expr*: Specify the number shape selects used to create this waveform. Default = no check.
- (10) **Shape**: This statement begins a block to specify the events that make up the shape and the timing limits of the waveform. Note that the syntax follows the same form as for a waveform as defined in STIL.0 clause 18. The difference between a STIL.0 waveform and this waveform is that only the event sequence is defined, and an optional sub-block may be attached to each event that defines the minimum and maximum time of the event. Note that the use of TRC\_LABEL and @labels allows for timing specification relative to other events. If absolute values are used for Min and Max time\_expr then they are relative to T0 of the period.

## 17.2 TRC - WaveformDescriptions - Example

# 18. TRC - PatternCharacteristics

## 18.1 TRC - PatternCharacteristics - Syntax

```

PatternCharacteristics (PAT_CHAR_NAME) { (1)
  ( Base < Hex | Dec | HexDec > INTEGER;) (2)
  ( BreakPoint < Yes | No >;) (3)
  ( ConditionalStatements < Yes | No > ;) (4)
  ( CoreUsageReady < Yes | No > ;) (5)
  ( NonCyclized < Yes | No >;) (6)
  ( DeltaChangeVectorData < Yes | No >;) (7)
  ( GoTo < Yes | No >;) (8)
  ( IddqTestPoints < Yes | No >;) (9)
  ( InstructionCharacteristics { (10)
    ( AllowedWhen 'boolean_expr'; )
    ( AppliesTo (< Condition | GoTo | Loop | MatchLoop | Call | Shift >)+ ;)
    ( LoopCharacteristics LOOP_NAME; )
    ( MinVectorsAfter integer_expr; )
    ( MinVectorsBefore integer_expr; )
    ( MinVectorsBetween integer_expr; )
  } )* // end InstructionCharacteristics
  ( LoopCharacteristics (LOOP_NAME) { (11)
    ( Infinite; )
  } )

```

```

    ( MaxIteration integer_expr; )
    ( MaxLength integer_expr; )
    ( MaxNest integer_expr; )
    ( MinIteration integer_expr; )
    ( MinLength integer_expr; )
    ( MinTimeAfterMatch real_expr; )
    ( MinVectorsAfterMatch integer_expr; )
  }* // end LoopCharacteristics
  ( Macro < Yes | No | WithParameters >; ) (12)
  ( MaxRunTime time_expr; ) (13)
  ( MaxVectors integer_expr; ) (14)
  ( MultiBitData (< InWaveforms | InPatterns | No >)* ; ) (15)
  ( NumberCaptureCycles (integer_expr (integer_expr)) ; ) (16)
  ( NumberPatternUnits integer_expr ; ) (17)
  ( NumberVectorsPerShift (integer_expr) (integer_expr); ) (18)
  ( PatternVariables < Yes | No > ; ) (19)
  ( ProcedureCalls < Yes | No | WithParameters > ; ) (20)
  ( Shift < Yes | No >; ) (21)
  ( STILPatterns < Yes | No >; ) (22)
} // end PatternCharacteristics

```

(1) **PatternCharacteristics** (PAT\_CHAR\_NAME): This block specifies the characteristics associated with running a pattern. This block may also be used to document the characteristics of a given pattern. The perspective of the descriptions herein is from the tester constraint perspective. Note that some of the statements have little or no significance as a tester hardware constraint and are so described.

(2) **Base** < **Hex** | **Dec** | **HexDec** > INTEGER: Specify whether vector data and parameters in radix other than wfc are supported. By default, only wfc data is supported. This pattern data characteristic is not typically a tester hardware constraint, but may be a constraint of the ATE software. The INTEGER defines the maximum number of wfc's that can be mapped from hex or decimal.

(3) **BreakPoint** < **Yes** | **No** >: Specify whether breakpoints are supported.

(4) **ConditionalStatements** < **Yes** | **No** >: Specify whether conditional statements are supported.

(5) **CoreUsageReady** < **Yes** | **No** >: Specify that patterns in core ready format are supported. Core ready requires that patterns contain Macro and Procedure calls only - i.e., no V or W statements in the pattern. This pattern data characteristic is not typically a tester hardware constraint, but may be a constraint of the ATE software.

(6) **NonCyclized** < **Yes** | **No** >: Specify whether patterns with non-cyclized data are supported.

(7) **DeltaChangeVectorData** < **Yes** | **No** >: Specify whether vectors and parameters in incremental format are supported. This pattern data characteristic is not typically a tester hardware constraint, but may be a constraint of the ATE software.

(8) **GoTo** < **Yes** | **No** >: Specify whether patterns containing GoTo and labels are supported.

(9) **IddqTestPoints** < **Yes** | **No** >: Specify whether patterns containing IddqTestPoint statements are supported.

(10) **InstructionCharacteristics**: This statement is used to specify the capabilities, limitations and characteristics of the various types of instructions that are allowed within a pattern. The following attribute statements are allowed within this block type. All statements are optional.

**AllowedWhen** *'boolean\_expr'*: This is a statement that can be used to specify a boolean expression that must evaluate to true for this instruction type. Typically this would be an evaluation based on the vector address. i.e., '*vector\_address%2*' would evaluate to true on every other address of the pattern.

**AppliesTo**: This statement is used to specify which pattern statements that the characteristics, as defined in this block, are to be associated with. The following are the allowed identifiers:

- (a) **Condition** - These characteristics are to be associated with the conditional pattern statements - If, While.
- (b) **GoTo** - These characteristics are to be associated with the GoTo pattern statement.
- (c) **Loop** - These characteristics are to be associated with the Loop pattern statement.
- (d) **MatchLoop** - These characteristics are to be associated with the MatchLoop pattern statement.
- (e) **Call** - These characteristics are to be associated with the Call pattern statement.
- (f) **Shift** - These characteristics are to be associated with the Shift pattern statement.

**LoopCharacteristics** *LOOP\_NAME* : This statement is to be used only on one of the looping statements (Loop, MatchLoop, Shift, BreakPoint) and refers to a specific named set of LoopCharacteristics. If this statement is not present, then the anonymous set of LoopCharacteristics is to be used.

**MinVectorsAfter** *integer\_expr* : This statement specifies the minimum vector (V) statements that must exist after to this instruction type with no intervening instructions.

**MinVectorsBefore** *integer\_expr* : This statement specifies the minimum vector (V) statements that must exist prior to this instruction type with no intervening instructions.

**MinVectorsBetween** *integer\_expr* : This statement specifies the minimum vector (V) statements that must exist between this instruction type and the next instruction with no intervening instructions.

(11) **LoopCharacteristics**: Specify the characteristics of the loop blocks. A loop may be specified by any of the following statements: Loop, MatchLoop, Shift, Breakpoint. There may be more than one of these blocks. e.g., the characteristics of a single vector repeat (loop of 1) may be different from a larger loop. The following are the allowed statements within this block:

- (a) **Infinite** - The loop may be infinite.
- (b) **MaxIteration** - Specify the maximum times the loop may be executed.
- (c) **MaxLength** - Specify the maximum number of Vectors that may occur within the loop.
- (d) **MaxNest** - Specify the maximum number of loops that may be contained within a loop. i.e., "MaxNest 8;" means that there may be an outer loop and up to 7 more loops embedded within.
- (e) **MinIteration** - Specify the minimum number of iterations that must be present in a loop. This would typically be one. A MinIteration of 0 indicates not to execute the loop at all. A MinIteration of 2 indicates that the loop must execute at least 2 times. Etc.
- (f) **MinLength** - Specify the minimum number of vectors that may occur within the loop.
- (g) **MinTimeAfterMatch** - Due to pipelining, an ATE system typically cannot respond with valid compare results immediately after a match is found. This statement specifies the minimum amount of time that is required before a compare can be done. See also MinVectorsAfterMatch.
- (h) **MinVectorsAfterMatch** - Due to pipelining, an ATE system typically cannot respond with valid compare results immediately after a match is found. This statement specifies the minimum number of vectors that are required before a compare can be done. See also MinTimeAfterMatch.

(12) **Macro** < **Yes** | **No** | **WithParameters** >: Specify whether patterns containing Macros are supported. This pattern data characteristic is not typically a tester hardware constraint, but may be a constraint of the ATE software.

(13) **MaxRunTime** *time\_expr*: Specify the maximum run time for a pattern execution. (default is no limit)

(14) **MaxVectors** *integer\_expr*: Specify the maximum number of vectors.

(15) **MultiBitData** (< **InWaveforms** | **InPatterns** | **No** >: Specify whether patterns with multibit vectors are supported.

- (a) **InWaveforms**: multiple bits used to select events in waveforms
- (b) **InPatterns**: multiple bits in pattern parameters to Macros and Procs

(16) **NumberCaptureCycles** (*integer\_expr* ( *integer\_expr*)): Specify the number of vectors in ATPG generated capture cycles. This pattern data characteristic is not typically a tester hardware constraint, but is used for documenting a pattern.

- (a) *no integers*: all capture sequences have same number of cycles
- (b) *first integer*: minimum number capture cycles
- (c) *second integer*: maximum number of capture cycles

(17) **NumberPatternUnits** *integer\_expr*: Specify the number of ATPG generated pattern sequences (i.e., scan patterns). This pattern data characteristic is not typically a tester hardware constraint, but is used for documenting a pattern.

(18) **NumberVectorsPerShift** (*integer\_expr*) ( *integer\_expr*): Specify the number of vectors in ATPG scan shift sequences. This pattern data characteristic is not typically a tester hardware constraint, but is used for documenting a pattern.

- (a) *no integers*: all shift sequences have same number of cycles
- (b) *first integer*: minimum number shift cycles
- (c) *second integer*: maximum number of shift cycles

(19) **PatternVariables** < **Yes** | **No** >: Specify whether variables for Macros, Procs, or expressions are supported.

(20) **ProcedureCalls** < **Yes** | **No** | **WithParameters** >: Specify whether patterns containing procedure calls are supported. Note that whereas Macros are typically supported by expanding the pattern statements in-line, procedures are typically supported by transferring control to a pre-loaded pattern sequence. Also note that it is possible to expand procedures to in-line pattern statements, however the rules for procedure calls shall be maintained.

(21) **Shift** < **Yes** | **No** >: Specify that shift is supported (typically used for Scan vectors). Shift may be done by either hardware or flattening into vectors.

(22) **STILPatterns** < **Yes** | **No** >: Specify whether the patterns are in fully STIL compliant format. This pattern data characteristic is not typically a tester hardware constraint.

## 18.2 TRC - PatternCharacteristics - Example

```

4: STIL 1.0 { Design D13; TRC D04; }
5: Header {
6:   Source "P1450.1 Draft 13, December 26, 2001";
7:   Ann {* clause 6.8 *}
8: } //end Header
9: Environment ate_foo {
10:  TRC {
11:   PatternCharacteristics {
12:     Base Hex 64;
13:     BreakPoint No;
14:     ConditionalStatements Yes;
15:     CoreUsageReady No;
16:     NonCyclized No;
17:     DeltaChangeVectorData Yes;

```

```

18:      GoTo No;
19:      IddqTestPoints Yes;
20:      InstructionCharacteristics {
21:        AppliesTo Condition GoTo Loop Call Shift;
22:        ( MinVectorsAfter 2;
23:        ( MinVectorsBefore 2;
24:        ( MinVectorsBetween 1;
25:      } // end InstructionCharacteristics
26:      InstructionCharacteristics {
27:        AppliesTo LoopMatch;
28:        ( MinVectorsAfter 2;
29:        ( MinVectorsBefore 2;
30:        ( MinVectorsBetween 35;
31:      } // end InstructionCharacteristics
32:      LoopCharacteristics {
33:        Match;
34:        MaxIteration 4096;
35:        MaxLength 65K;
36:        MaxNest 3;
37:        MinIteration 2;
38:        MinLength 2;
39:        MinNest 1;
40:      } // end LoopCharacteristics
41:      Macro WithParameters;
42:      MaxRunTime `15s`;
43:      MaxVectors 5M;
44:      MultiBitData InWaveforms;
45:      MultipleSites 4;
46:      MultiplePorts 2;
47:      PatternVariables No;
48:      ProcedureCalls Yes;
49:      Shift Yes;
50:      STILPatterns Yes;
51:    } // end PatternCharacteristics
52:  } // end TRC
53:} // end Environment

```

## 19. TRC - NameChecks block

STIL defines a specific set of rules for naming objects, and identifies namespaces that contain each type of object. Different environments and applications of STIL data often define different naming constructs from the STIL environment.

When STIL names are passed to contexts outside of STIL, additional restrictions may be imposed on these names. The NameChecks block is used to validate STIL names against a limited set of additional naming restrictions. These limited checks support checking names against an environment that supports reduced naming flexibility. It may be necessary in some circumstances to define restrictions using these rules that are a subset of the complete external context in order to simplify the checks to a naming environment that overlaps both the basic STIL constructs and the external environment. Obviously a name in an external context cannot be defined that is more flexible than the incoming STIL capabilities.

NameChecks operations come in three forms: character-content, length, and scope. They are specified against the names defined either inside a STIL block (for instance, the contents of the STIL Signals block), or against the blocknames of a STIL block (for instance, across the names of all WaveformTable blocks). Character-content checks validate the set of characters contained in a name are appropriate; length checks confirm restrictions on the length of a name; and scope constraints check that this name is uniquely defined across the specified set of STIL blocks.

NameChecks shall be performed under an executable STIL context, that is, after STIL name resolution processes have identified a set of names that are actually used under a PatternExec context.

## 19.1 NameChecks block - Syntax

```

NameChecks NAM_CHECKS_NAME { (1)
  ( Contents STIL_BLOCK_NAME ; ) * (2)
  ( Block STIL_BLOCK_NAME ; ) * (3)
  ( CharacterContent 'regular_expr' ; ) (4)
  ( Length integer_expr ; ) (5)
  ( Scope (STIL_BLOCK_NAME)+ ; ) (6)
}

```

(1) **NameChecks** NAM\_CHECKS\_NAME is an optional STIL block inside TesterResourcesConstraints to identify a set of STIL naming restrictions for this TesterResourcesConstraints block. NAM\_CHECKS\_NAME shall be unique across all NameChecks blocks in one TesterResourcesConstraints block.

(2) **Contents** STIL\_BLOCK\_NAME defines a set of STIL names, defined by user-defined keyword statements, in this STIL\_BLOCK\_NAME, to be checked against the CharacterContent, Length, and Scope requirements identified in this block. STIL\_BLOCK\_NAME is any STIL block that contains user-defined name keywords (such as Signals). At least one Contents statement or Block statement shall be present in a NameChecks block. If the keyword **AllNames** is specified for STIL\_BLOCK\_NAME, then All STIL namespaces will be checked against these constraints.

(3) **Block** STIL\_BLOCK\_NAME defines a set of STIL names, defined by the set of all blocknames defined under STIL\_BLOCK\_NAME, to be checked against the CharacterContent, Length, and Scope requirements identified in this block. STIL\_BLOCK\_NAME is any STIL block. At least one Contents statement or Block statement shall be present in a NameChecks block

(4) **CharacterContent** 'regular\_expr' defines a regular expression construct to identify any character restrictions on a name.

*TBD: Need a reference to some standard that defines what a "regular expression" is.*

(5) **Length** integer\_expr defines an absolute limit on the number of characters a name may contain.

(6) **Scope** (STIL\_BLOCK\_NAME)+ defines one or more STIL blocks to check for a unique name. This statement is only required when the TesterResourcesConstraints namespace context is different than the STIL environment, typically containing multiple namespaces. If the keyword **AllNames** is specified for STIL\_BLOCK\_NAME, then each name checked here will be checked against all the names contained in all STIL namespaces. Be aware that STIL namespaces that are defined unique under another name (for instance, ScanChain names inside a ScanStructures block) will not maintain that restriction if that block is referenced in a Scope statement.

## 19.2 NameChecks Examples

54 :// partial WGL name checks for Signals, Groups and ScanChains

```

55:TesterResourceConstraints WGL {
56:  NameChecks for_signals {
57:    // in WGL, Signals and SignalGroups share the same namespace
58:    Contents Signals;
59:    Contents SignalGroups;
60:    CharacterContent '(\".*\\"|[A-Za-z][A-Za-z0-9_\-]*)';
61:    // Names are either enclosed in double-quotes,
62:    // or start with an alphabetic character, followed by
63:    // any number of alphanumeric characters, underscore, or dash.
64:    // Does not trap use of WGL reserved words
65:    Scope Signals SignalGroups;
66:  }
67:  NameChecks for_schainchains {
68:    Block ScanChain;
69:    CharacterContent '(\".*\\"|[A-Za-z][A-Za-z0-9_\-]*)';
70:    // Does not trap use of WGL reserved words
71:    Scope ScanChains;
72:    // Note STIL ScanChain names are scoped under ScanStructures.
73:    // This statement assures unique names for all chains across
74:    // all referenced ScanStructure blocks.
75:  }
76:}
77:// Namechecks for a tester that places all names into one big, untyped, pool.
78:TesterResourceConstraints flat_tester {
79:  NameChecks for_all_names {
80:    // This context supports only one big namespace
81:    Contents AllNames;
82:    Length 22;
83:    Scope AllNames;
84:  }
85:}

```

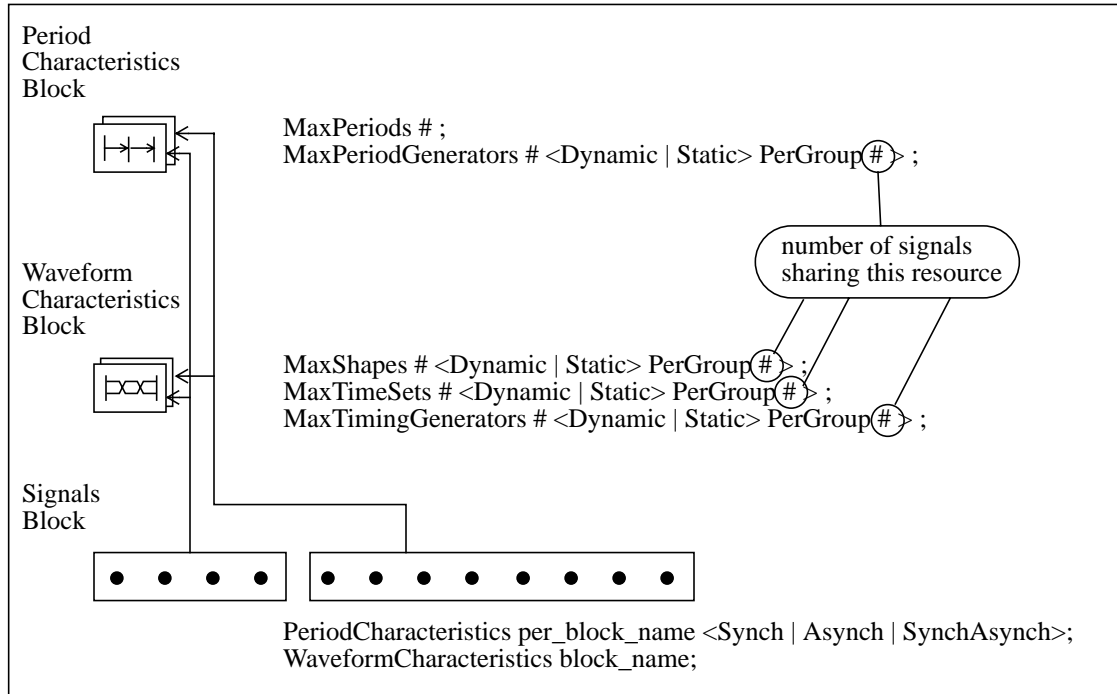
## 20. Waveform Generator Model

(informative)

The waveform generation architecture of an ATE system can take on many different forms. The rules are defined to be flexible enough to describe commonly used architectures from waveform generation-per-pin to

a central timing architecture. The diagram below shows the generic picture of an ATE system and how the constraint rules apply.

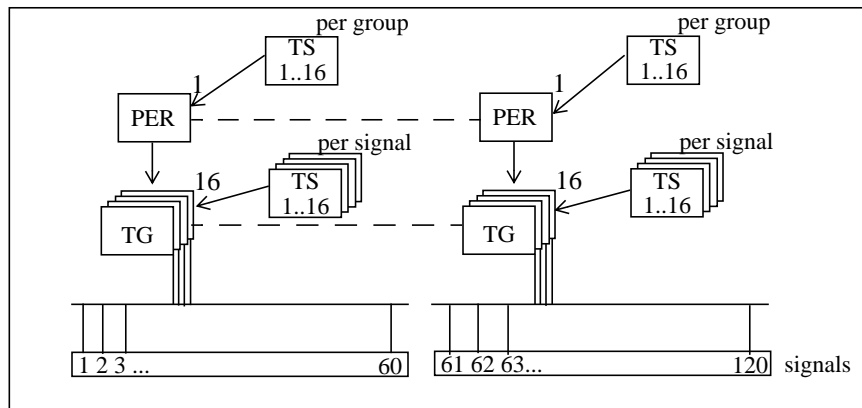
Figure 2: General Timing Model



To better illustrate the timing resource assignment, the following example shows a system with the following characteristics:

- 120 signals arranged in two groups of 60.
- 1 period generator for each group of 60 signals. Each period generator has 16 values that are selectable by the per signal time set select.
- 16 timing generators can be assigned to any of the signals statically. Each TG has 16 values that are selectable by the per signal time set select.
- 16 time sets for each signal that can be selected dynamically

Figure 3: Example of Timing Model with Static Timing Select

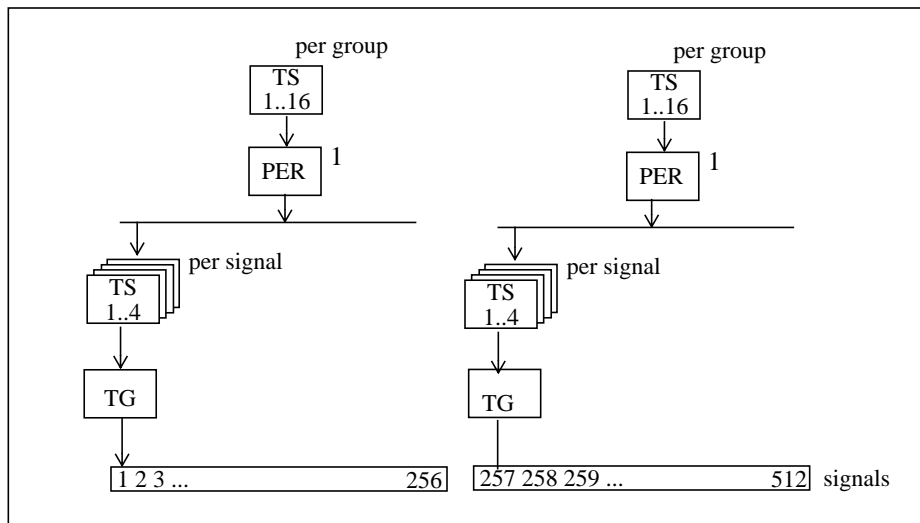


Here is the code that defines the above architecture. Note: see Annex Annex A: for complete test constraints for this archetecture.

```

86:Environment {
87:  TestResourceConstraint ATE_X {
88:    PeriodCharacteristics PER {
89:      MaxPeriods 16 Dynamic PerGroup 120;
90:      MaxPeriodGenerators 1 Static PerGroup 120;
91:    }
92:    WaveformCharacteristics WAV {
93:      FormatSelect InOut {
94:        MaxTimeSets 16 Dynamic PerSignal;
95:        MaxTimingGenerators 16 Static PerGroup 120;
96:      }
97:    }
98:    SignalCharacteristics Data Clock In Out {
99:      MaxSignals 120;
100:     PeriodCharacteristics PER Synch;
101:     WaveformCharacteristics WAV;
102:   }
103: }
104:}
    
```

Figure 4: Example of Timing Model with Per Signal Timing



Here is the code that defines the above architecture.

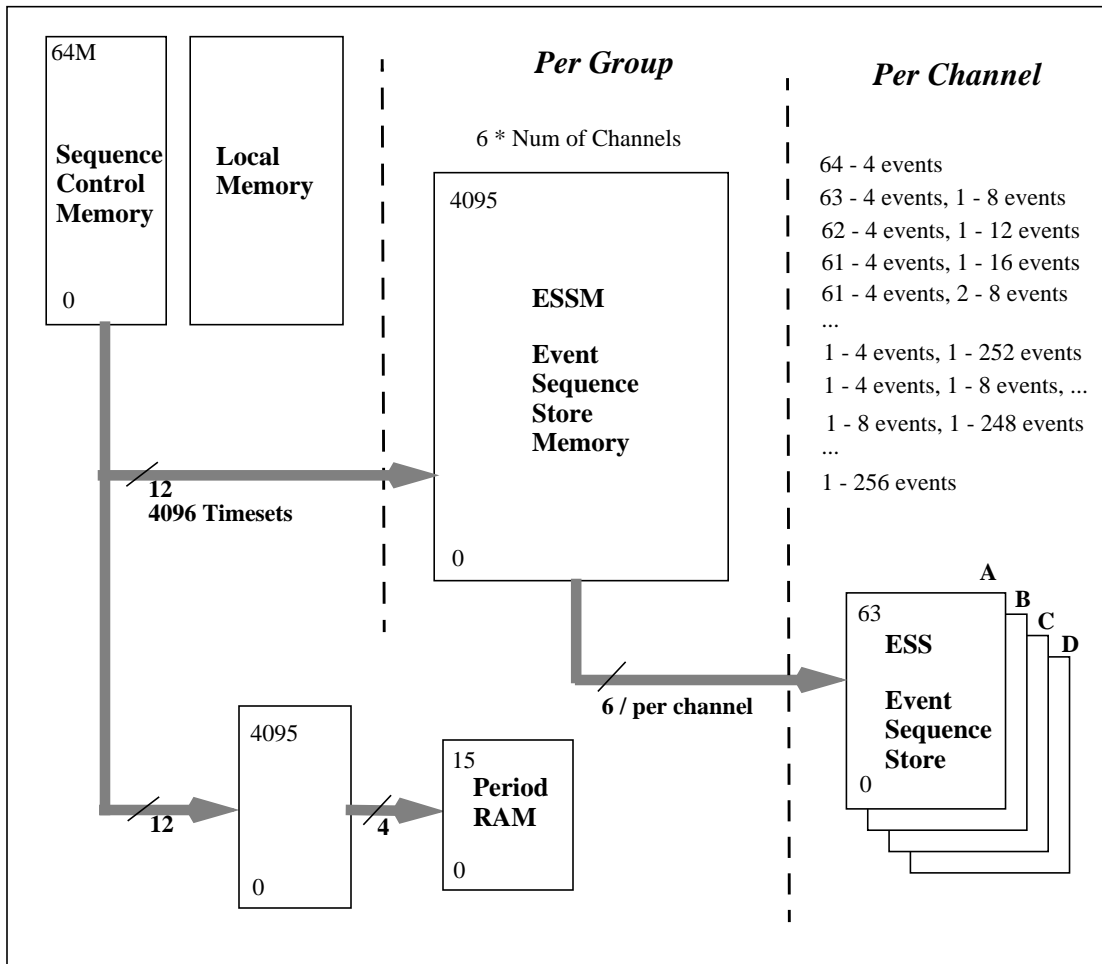
```

105:Environment {
106:  TestResourceConstraint ATE_Y {
107:    PeriodCharacteristics PER {
108:      MaxPeriods 16 Dynamic PerGroup 256;
109:      MaxPeriodGenerators 2 Static PerGroup 256;
110:    }
111:    WaveformCharacteristics WAV {
112:      FormatSelect InOut {
113:        MaxTimeSets 4 Dynamic;
    
```

```

114:    }
115:  }
116:  SignalCharacteristics Data Clock In Out {
117:    MaxSignals 1024;
118:    PeriodCharacteristics PER SynchAsynch;
119:    WaveformCharacteristics WAV;
120:  }
121: }
122:}
    
```

Figure 5: Example of Timing Model with Waveform Select Memory



The following code shows is for the above architecture with select memory:

```

123:Environment {
124:  TestResourceConstraint ATE_Z {
125:    PeriodCharacteristics PER {
126:      MaxPeriods 16 Dynamic PerGroup NUM_PINS;
127:      MaxPeriodGenerators 1 Static PerGroup NUM_PINS;
128:      PeriodSelectMemory 4096 PerGroup NUM_PINS;
129:    }
    }
    }
    
```

```

130:   WaveformCharacteristics WAV {
131:     FormatSelect InOut {
132:       MaxTimeSets 64 Dynamic;
133:       MaxTimingGenerators 1 PerSignal;
134:       WaveformSelectMemory 4096 PerGroup NUM_PINS;
135:       SelectWithPeriod;
136:     }
137:   }
138:   SignalCharacteristics Data Clock In Out {
139:     PeriodCharacteristics PER SynchAsynch;
140:     WaveformCharacteristics WAV;
141:     MaxSignals NUM_PINS;
142:   }
143: }
144:}

```

## 21. Fluid Concepts in Parameter Specification

(informative)

In many cases a given ATE system can be configured in many ways with optional or modular additions to the hardware of a system. This flexibility is referred to in this document as a “fluid” specification, and is handled by means of Spec variables.

The definition of category names and variable names is contained within a given STIL file, however, for the TestResourceConstraints application, the following guidelines should be followed. There are two types of constraints that may be defined. a) There are fixed characteristics of a tester that relate to a specific model of tester such as max pin size, hardware options ranges available. These should be defined as Spec->Category names. b) Then there are things that are selectable by a given application. These should be defined as Spec->Var names. Keep in mind that it is possible to create a variable expression that includes other variable names, but it is not permissible to reference another category.

The following sections illustrate examples of fluid specification.

### 21.1 Example of Pins in a Segment

This is an example of an ATE system that has variable pin assignments. There are two configurations of the tester defined, one with a maximum of 1024 pins and the other with 512 pins. In both cases, the tester may have either 1 or 2 segments with the pins being assigned to the segments in increments of 64. The following CTL/STIL code defines this case:

```

145:STIL 1.0 { CTL 2001; }
146:
147:Spec vars_for_test_constraints {
148: Category size_1024 {
149:   max_pins = 1024;
150:   seg_one_pins = `max_pins mod 64;
151:   seg_two_pins = `max_pins - seg_one_pins;
152: }
153: Category size_512 {
154:   max_pins = 512;

```

```

155:   seg_one_pins = `max_pins mod 64;
156:   seg_two_pins = `max_pins - seg_one_pins;}
157:} // end Spec
158:
159:Environment tester_specs { CTL {
160: TestResourceConstraints model_s {
161:   Category size_1024 size_512;
162:   Segment seg_one {
163:     MaxSegments 1;
164:     Pins data {
165:       MaxPins `seg_one_pins';
166:     }
167:   }
168:   Segment seg_two {
169:     MaxSegments 1;
170:     Pins data {
171:       MaxPins `seg_two_pins';
172:     }
173:   } // end Segment
174: } // end TestResourceConstraints
175:}} // end Environment, CTL
176:

```

## 21.2 Example of Multiple Ranges on the Period

This example of fluid ATE constraints is defining allowed ranges on the period specification. The ATE system has the ability to switch between 16 different period values on-the-fly. Each of these periods must select from one of three ranges based on the value that is to be programmed into the period. Once the period range is selected, the specification for accuracy, resolution, and max edge times are determined from the period. The following shows how this is specified by means of three category names - range1, range2, and range3.

```

177:STIL 1.0 { Design D14; CTL 2001; }
178:Spec vars_for_test_constraints {
179: Category range1 {
180:   per = `100ns';
181:   acc = `200ps';
182:   res = `100ps';
183: }
184: Category range2 {
185:   per = `1us';
186:   acc = `5ns';
187:   res = `1ns';
188: }
189: Category range3 {
190:   per = `10us';
191:   acc = `per/1000';
192:   res = `per/2000';
193: }
194:} //end Spec
195:Environment tester_specs { CTL {
196: TestResourceConstraints model_t {
197:   Category range1 range2 range3;
198:   Segment {

```

```
199:     Periods {
200:         MaxPeriods 16; // max ranges switched on the fly
201:         MinPeriodTime `10ns` ; // fixed, regardless of range
202:         MaxPeriodTime `per` ;
203:         Resolution `res` ;
204:         Accuracy `acc` ;
205:     } // end Periods
206:     WaveformCharacteristics waveform_info_name {
207:         MaxEdgeTime `(2*per)-10ns` ; // max time from T0 to edge
208:         Resolution `per/4096` ;
209:         Accuracy `per/8192` ;
210:     } ) * // end WaveformCharacteristics
211: } // end Segment
212: } // end TestResourceConstraints
213: }} // end of CTL, Environment
```

## Annex A: CTL Description of a distributed-resource Tester

(informative)

January 23, 2002 - from Dan Fan - S21

This is a simplified model of a distributed-resource Tester. It is a 20MHz tester with distributed timing system of 16 Timing Generators, 16 Timing Sets, 4K standard vector memory (Local Memory) with optional Extended Local Memory of 1Meg vector depth. The timing format are fixed as NRZ, RTZ, RTO, and SBC (Surrounding-By-Complement, i.e. XOR).

The following, then, is the CTL description of this tester:

```

214:Spec DistributedTester_Vars {
215:  Category DistributedTester_Category {
216:    PeriodVal {Min='50ns', Max = '4ms'}
217:  } // end Category
218:} // end Spec
219:
220:TRC DistributedTester_Rules {
221:  // DistributedTester is a tester which is a distributed resource tester with limited channels (120).
222:  Category DistributedTester_Category;
223:  PeriodCharacteristics PeriodInfo {
224:    Accuracy '1ns'; // DistributedTester's period accuracy is around 1ns
225:    MaxPeriods 16; // DistributedTester has 16 Period Selections
226:    MaxPeriodGenerators 1; // DistributedTester only has one system period for all channels
227:    PeriodTimeLimit PeriodVal;
228:    Resolution '5ns'; // Period resolution
229:  } // end PeriodCharacteristics
230:
231:  WaveformCharacteristics TimingInfo {
232:    Accuracy Edge '500ps';
233:    CompareEvents Window H/L/X X 1;
234:    DriveEvents U/D U/D 1;
235:    FormatSelect In {
236:      MaxShape 4 Static;
237:      // Input has 4 possible formats (NRZ, RTZ, RTO, SBC), single selected Format per run
238:      MaxTimeSets 16 Dynamic { PerTimingGenerator; }
239:      // 16 Time Sets (TS)
240:      MaxTimingGenerators 12 Static;
241:      // TG1A/B, TG2A/B to TG6A/B, total of 12 TGs for inputs
242:      MaxData Drive 2 Dynamic;
243:      // Data can be either 0 or 1
244:      MaxIO 2 Dynamic { PerGroup 16; }
245:      // 16 D registers, each register defines which signal is input or output;
246:      // these D register can be selected per vector
247:    } // end FormatSelect
248:    FormatSelect Out {
249:      MaxShape 1 Static;
250:      // Output strobe can only support as window strobe format
251:      MaxTimeSets 16 Dynamic { PerTimingGenerator; }
252:      // 16 Time Sets (TS)
253:      MaxTimingGenerators 4 Static;

```

```

254:      //TG7A/B to TG8A/B, total of 4 TGs for outputs
255:      MaxData Compare 2 Dynamic;
256:      // Data can be either 0 or 1
257:      MaxIO 2 Dynamic { PerGroup 16; }
258:      // 16 D registers, each register defines which signal is input or output;
259:      // these D register can be selected per vector
260:      MaxMask 2 Dynamic { PerGroup 16; }
261:      // 16 Mask registers, each register defines which signal is masked or not;
262:      // these Mask registers can be selected per vector
263:    } // end FormatSelect
264:    MaxEdgeTime '2 * Period_val - 20ns';
265:    MaxEvents Compare 2; // window strobe needs 2 events
266:    MinCompareWindow '3ns';
267:    MinDrivePulse '3ns';
268:    Resolution '156ps';
269:  } // End WaveformCharacteristic
270:
271:  WaveformDescriptions TimingFormat Explicit {
272:    In NRZ_Format {
273:      NumberData 2;
274:      NumberIO 2;
275:      NumberPeriods 1;
276:      NumberShapes 1;
277:      NumberSignals 1;
278:      NumberTimeSets 1;
279:      Shape {
280:        NRZ_E1: U/D { Max '2 * Period_val - 20ns';}
281:      } // end Shape
282:    } // end In
283:    In RTZ_Format {
284:      NumberData 2;
285:      NumberIO 2;
286:      NumberPeriods 1;
287:      NumberShapes 1;
288:      NumberSignals 1;
289:      NumberTimeSets 1;
290:      Shape {
291:        U/D;
292:        RTZ_E2: D { Min '@+3ns'; Max '2 * Period_val - 20ns';}
293:      } // end Shape
294:    } // end In
295:    In RTO_Format {
296:      NumberData 2;
297:      NumberIO 2;
298:      NumberPeriods 1;
299:      NumberShapes 1;
300:      NumberSignals 1;
301:      NumberTimeSets 1;
302:      Shape {
303:        U/D;
304:        RTO_E2: U { Min '@+3ns'; Max '2 * Period_val - 20ns'; }
305:      } // end Shape
306:    } // end In
307:    In SBC_Format {

```

```

308:    NumberData 2;
309:    NumberIO 2;
310:    NumberPeriods 1;
311:    NumberShapes 1;
312:    NumberSignals 1;
313:    NumberTimeSets 1;
314:    Shape {
315:        '0ns' U/D;
316:        SBC_E1: D/U { Min '@+3ns; Max '2 * Period_val-20ns'; };
317:        SBC_E2: U/D { Min '@+3ns'; Max '2 * Period_val - 20ns'; };
318:    } // end Shape
319: } // end In
320: Out Compare_Format {
321:    NumberData 2;
322:    NumberIO 2;
323:    NumberMask 2; // controlled by M register 0 or 1
324:    NumberPeriods 1;
325:    NumberShapes 1;
326:    NumberSignals 1;
327:    NumberTimeSets 1;
328:    Shape {
329:        L/H;
330:        Compare_E2: X { Min '@+3ns'; Max '2 * Period_val - 20ns'; };
331:    } // end Shape
332: } // end Out
333: } // end WaveformDescriptions
334:
335: SignalCharacteristics Data Clock Scan In Out {
336:    InOut OnCycleBoundary;
337:    MaxSignals 120;
338:    NumberVectors 4096;
339:    PeriodCharacteristics PeriodInfo Synch;
340:    WaveformCharacteristics TimingInfo;
341:    WaveformDescriptions TimingFormat;
342: } // end SignalCharacteristics
343:
344: PatternCharacteristics {
345:    DeltaChangeVectorData No;
346:    MaxLoop 16 Match;
347:    NumberVectors 4096;
348:    VectorRepeat Yes 1024;
349: } // end PatternCharacteristics
350: } // End TRC

```

## ***Annex B: CTL Description of a structural tester***

(informative)

*January 23, 2002, from Dan Fan, DeFT*

This is a simplified model of a structural tester. It consists of 4 PLL clocks, 160 Signals of regular data channels.

The 4 PLL clock Signals have 2 drive edges in each clock period and the clock period is separated from vector period. The clocks have fixed format of either RTZ or RTO with 50% duty cycle. The vector period is up to 200MHz (5ns)

The data channels are single event per vector period. The regular vector memory has 16 million vectors and 2K subroutine memory. The separated scan memory has 256 million vectors. Within the 160 data channels, 96 channels are supported as Scan In/Out.

The following, then, is the CTL description of this tester:

```

351:Spec StructuralTester_vars {
352: Category StructuralTester_category {
353:   PeriodVal {Min '5ns', Max '4us'}
354:   NumScanOutPins {Min '0', Max '48'}
355:   NumScanInPins {Min '0', Max '96-NumScanOutPins'}
356: } //end Category
357:} //end Spec
358:
359:TRC StructuralTester_rules {
360: //This is a Structural Tester which is based on the Sequence-Per-Pin architecture;
361: //In this spec, it is configured as 160 test channels.
362: Category StructuralTester_category;
363: PeriodCharacteristics PeriodInfo {
364:   Accuracy '10ps';
365:   MaxPeriods 1;
366:   MaxPeriodGenerators 1;
367:   PeriodTimeLimit PeriodVal;
368:   Resolution '10ps';
369: } //end PeriodCharacteristics
370:
371: WaveformCharacteristics ClockWav {
372:   Accuracy Edge '150ps';
373:   Accuracy EdgetoEdge '300ps';
374:   DriveEvents U/D 1;
375:   FormatSelect In {
376:     MaxShapes 2 Static; //Clock can be RTZ or RTO format
377:     MaxData Drive 2 Dynamic;
378:     MaxTimeSets 64 Dynamic { PerGroup 1024; }
379:   } //end FormatSelect
380:   MaxEvents Drive 2 ;
381:   Resolution '10ps';
382: } //End WaveformCharacteristics ClockWav
383:
384: WaveformCharacteristics DataWav {

```

```

385: Accuracy Edge '500ps';
386: Accuracy EdgetoEdge '1ns';
387: CompareEvents Edge H/L/X 1;
388: DriveEvents U/D/Z 1;
389: FormatSelect InOut {
390:   MaxShapes 64 Dynamic;
391:   MaxData DriveCompare 2 Dynamic;
392:   MaxTimeSets 64 Dynamic { PerGroup 1024; }
393: } // end Formats DataWav
394: MaxEdgeTime '4 * PeriodVal';
395: MaxEvents Drive 1;
396: MaxEvents Compare 1;
397: Resolution '40ps';
398: } // End Data Waveform Characteristics
399:
400: WaveformDescriptions ClockFormat Explicit {
401:   In RTZ_Format {
402:     Shape {
403:       U/D;
404:       RTZ_E2: D { MinMax '@+PeriodVal/2';}
405:     } // end Shape
406:   } // end In RTZ_Format
407:   In RTO_Format {
408:     Shape {
409:       U/D;
410:       RTO_E2: U { MinMax '@+PeriodVal/2';}
411:     } // end Shape
412:   } // end In RTO Format
413: } // end WaveformDescriptions
414:
415: SignalCharacteristics Clock {
416:   MaxSignals 4;
417:   DriveState U D;
418:   InOut Static;
419:   MaxVectorMemory '16*1024*1024';
420:   PeriodCharacteristics PeriodInfo Synch;
421:   WaveformCharacteristics ClockWav;
422:   WaveformDescriptions ClockFormat;
423: } // end SignalCharacteristics
424:
425: SignalCharacteristics Scan Out {
426:   MaxSignals NumScanOutPins;
427:   CompareStrobe Edge;
428:   CompareState H L X;
429:   MaxScanMemory '256*1024*1024';
430:   MaxVectorMemory '16*1024*1024';
431:   PeriodCharacteristics PeriodInfo Synch;
432:   WaveformCharacteristics DataWav;
433: } // end SignalCharacteristics
434:
435: SignalCharacteristics Scan In {
436:   MaxSignals NumScanInPins;
437:   DriveState U D;
438:   MaxScanMemory '256*1024*1024';

```

```
439:   MaxVectorMemory '16*1024*1024';
440:   PeriodCharacteristics PeriodInfo Synch;
441:   WaveformCharacteristics DataWav;
442: } // end SignalCharacteristics
443:
444: SignalCharacteristics Data In Out {
445:   MaxSignals '160-NumScanOutPins-NumScanInPins';
446:   CompareStrobe Edge;
447:   CompareState H L X;
448:   DriveState U D Z;
449:   InOut WithinCycle;
450:   MaxVectorMemory '16*1024*1024';
451:   PeriodCharacteristics PeriodInfo Synch;
452:   WaveformCharacteristics DataWav;
453: } // end SignalCharacteristics
454:
455: PatternCharacteristics {
456:   MaxLoop 4096;
457:   MaxRepeat '128*1024';
458:   MaxLoopNest 8;
459:   NumberVectorsPerShift '128*1024';
460: } // end PatternCharacteristics
461: } // End TRC
```

## Annex C: Test Resourse Constraints Example - SC212

(informative)

From - Frances Miller/Fluence (11/2/01), SC212

Tester characteristics:

- 6 bits per pin are assigned to the full depth of memory (1.5M) as follows:
  - 2 bits for drive control
    - drive high/low
    - drive on/off
  - 2 bits compare control
    - compare high/low/tri-state/dont'care
  - 2 bits time set control
    - 4 timesets switchable on the fly
    - SBC formats use 2 timesets
- There are 4 periods available. Only 1 can be selected at any given time across all pins.
- There are 2 edges available for drive (DNRZ, RZ, RO), 2 for compare, and 2 for direction.
- The SBC format is also supported. It takes up 2 timesets.
- A multi-clock feature is available on each pin.

```

462:Spec SC212_Vars {
463:  Category Model_A {
464:    Period_Value {Min `20ns`, Max `40960ns`}
465:    MultiValue {Min `1`, Max `100`}
466:  }
467:
468:TestConstraints SC212_FSM {
469:
470:  Category Model_A;
471:
472:  PeriodCharacteristics Standard_IO {
473:    MaxPeriods 4;
474:    MaxPeriodGenerators 1;
475:    PeriodTimeLimit Period_value;
476:    Resolution `.02 * Period_Value`;
477:  }
478:
479:  WaveformCharacteristics IO_Card {
480:    CompareEvents H/L/X/T 1;
481:    CompareStrobe Edge;
482:    CompareStrobe Window;
483:    DriveEvents P U D U/D/Z 1;
484:    FormatSelect In Out InOut { //pattern memory resources
485:      MaxShapes 4 Dynamic; //2 bits to select shape/timeset
486:      MaxTimeSets 1;//no additional timeset memory
487:      MaxData Drive 3 Compare 4;//2 bits to select level 0,1,Z
488:      //2 bits to select expect levels 0,1,Z,X

```

```
489:     }
490:     MaxEvents 2 Drive 2 Compare 2 OnOff; //2 edges for Drive, 2 for
491:         // compare, 2 for OnOff
492:     ReplicateSubWaveform 'MultiValue';
493:     MinEdgeReTrigger 14ns;
494:     MinDrivePulse 5ns;
495:     MinCompareWindow '5ns';
496:     Resolution 10ps;
497: }
498:
499: WaveformDescriptions IO_Card {
500:     SBC {
501:         Signals 1;
502:         NumberShapes 2; //uses up 2 shape selections
503:         Shape {
504:             U/D/Z
505:             D/U/Z {Max '@+6.2ns';}
506:             U/D/Z {Max 'PeriodValue - 6.2ns';}
507:         }
508:     }
509: }
510:
511: SignalCharacteristics SplitIO {
512:     InOut WithinCyle OnCycleBoundary;
513:     MaxSignals 304;
514:     MaxVectorMemory 4194304;
515:     MaxScanMemory 134217728;
516:     PeriodCharacterisitics Standard_IO Synch;
517:     WaveformCharacteristics IO_Card;
518:     WaveformDescriptions IO_Card;
519: }
520:
521: PatternCharacteristics {
522:     GoTo Yes;
523:     MaxLoop 65536 Match;
524:     MaxLoopNest 0;
525:     VectorRepeat Yes 65536;
526: }
527: }
```

## **Annex D: Test Resourse Constraints Example - Agilent**

(informative)

*TBD: xxx - Ken Posse, Agilent*

## Annex E: TestResourceConstraints Example

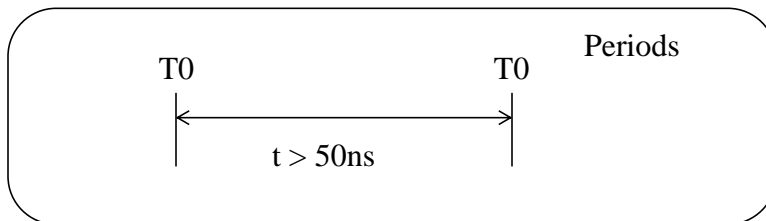
(informative)

*TBD: The following example needs work. Was taken from Toshiba definition of test constraint requirements. (tony)*

```

528:STIL 1.0 {CTL 2001; Design D14;}
529:Environment TesterRules {
530:  CTL {
531:    TestResourceConstraints T3320 {
532:      Periods {
533:        MaxPeriods 1;
534:        MinPeriod `50ns`;
535:        Resolution `1ns`;
536:      }

```

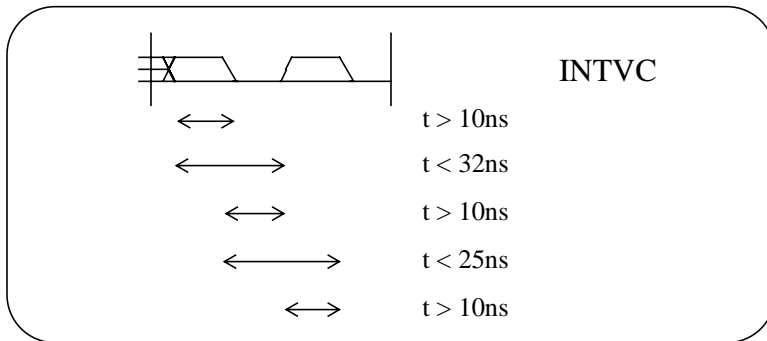


```

537:  Pins Data {
538:    MaxPins 256;
539:    MaxTimeSets 12 4;
540:    Resolution `1ns`;
541:    Waveforms {
542:      INTVC { // two cycle double pulse, INTVC, TWMIN
543:        Shape {
544:          INTVC_E1: U/D/P;
545:          INTVC_E2: D;
546:          INTVC_E3: U/D;
547:          INTVC_E4: D;
548:        }
549:        Checks {
550:          MinCheck INTVC_E2 `@+10ns`;
551:          MaxCheck INTVC_E3 `@1+32ns)`;
552:          MinCheck INTVC_E3 `@2+10ns)`;
553:          MaxCheck INTVC_E4 `@2+25ns)`;
554:          MinCheck INTVC_E4 `@3+10ns)`;
555:        }

```

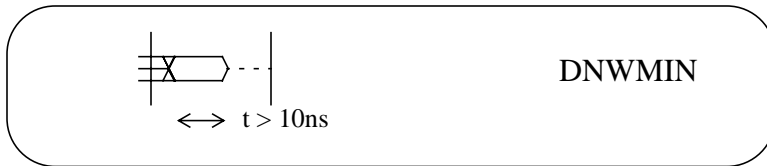
556:            }



```

557:            DNWMIN {    // delayed non-return to zero, DNWMIN
558:                Shape {
559:                    DNWMIN_E1: U/D/P;
560:                    DNWMIN_E2: Z;
561:                }
562:                Checks {
563:                    DNWMIN_C1: MinCheck DNMIN_E2 '@+10ns';
564:                }
              }

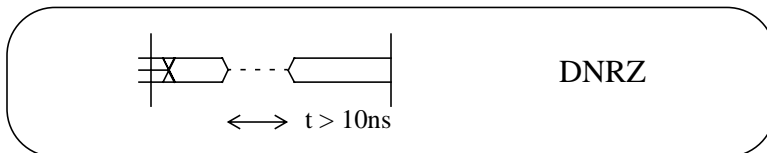
```



```

565:            DNRZ {    // DNRZ Bidir, INTVD
566:                Shape {
567:                    U/D/P;
568:                    Z;
569:                    DNRZ_E3: U/D;
570:                }
571:                Checks {
572:                    MinCheck DNRZ_E3 '@+10ns)';
573:                }
574:            }

```



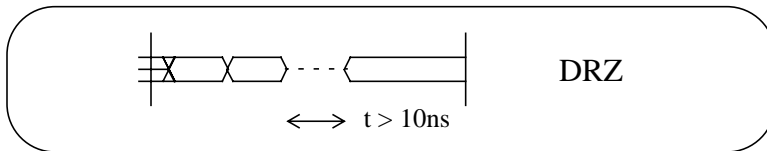
```

575:            DRZ {    // DRZ Bidir, INTVD
576:                Shape {
577:                    U/D/P;
578:                    U/D;
579:                    Z;
580:                    DRZ_E4: U/D/P;
581:                }

```

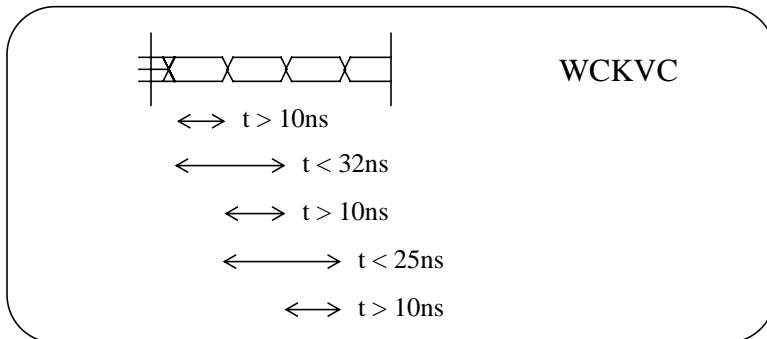
```

582:         Checks {
583:           MinCheck DRZ_E4: '@+10ns';
584:         }
585:       }
    
```



```

586:         WCKVC { // single cycle double pulse, WCKVC
587:           Shape {
588:             U/D/P;
589:             WKVC_E2: U/D;
590:             WKVC_E3: U/D;
591:             WKVC_E4: U/D;
592:           }
593:           Checks {
594:             MinCheck WKVC_E2: '@+10ns';
595:             MaxCheck WKVC_E3 '@1+32ns';
596:             MinCheck WKVC_E3 '@2+10ns';
597:             MaxCheck WKVC_E4 '@2+25ns';
598:             MinCheck WKVC_E4 '@3+10ns';
599:           }
600:         }
    
```

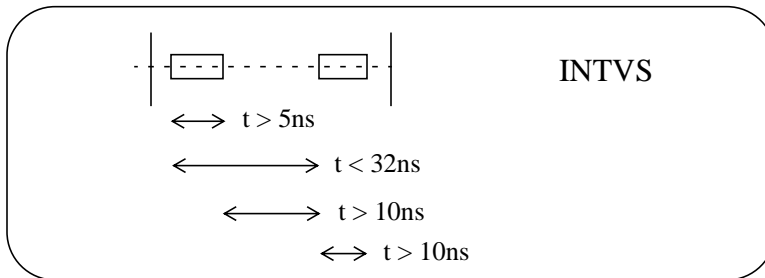


```

601:         RZ { // return to zero
602:           Shape {
603:             U/D/P;
604:             U/D;
605:           }
606:         }
607:
608:         INTVS { // window compare, INTVS
609:           Shape {
610:             L/H;
611:             INTVS_E2: X;
612:             INTVS_E3: L/H;
613:             INTVS_E4: X;
614:           }
615:           Checks {
616:             MinCheck INTVS_E2 '@+5ns';
617:             MaxCheck INTVS_E3 '@1+32ns';
    
```

```

618:          MinCheck INTVS_E3 '@2+10ns';
619:          MinCheck INTVS_E4 '@+5ns';
620:        }
621:    }
    
```



```

622:    } // end of Waveforms
623:
624: } // end of Pins Data
625:
626: Vectors {
627:     MaxVectors 65536 '200ns';
628:     MaxVectors 32768 '100ns';
629:     MaxVectors 16384 '50ns';
630: }
631: } // end of TestResourceConstraints T3320
632: } // end of CTL
633:} // end Environment
    
```

## Annex F: Example of Implicit Resource Assignment

(informative)

This annex is to illustrate an alternate method of identifying shared resource assignments within a timing block. Please compare and contrast this to the usage of the <resource id> labels which accomplish a similar purpose. See “Example of Resource Assignment in a Timing Block” on page 18. The difference between these two approaches is that the <resource\_id> labels can be added in a post processing flow, whereas the inheritance technique illustrated in this annex is part of the originally generated code.

```

634:STIL 1.0 { Design D13; TRC D04; }
635:Header {
636: Source "P1450.1 Draft 13, December 26, 2001";
637: Ann {* clause F *}
638:} //end Header
639:
640:Timing basic {
641:   WaveformTable period_one {
642:     Period `500ns`;
643:   }
644:   WaveformTable period_two {
645:     Period `550ns`;
646:   }
647:   WaveformTable period_three {
648:     Period `550ns`;
649:   }
650:   WaveformTable sequence_one {
651:     Waveforms {
652:       DIR { 01 { `0ns` D/U }}
653:       OE_ { 01 { `0ns` U; `200ns` D/U; `300ns` U; }}
654:       ABUS { 01 { `10ns` D/U; `300ns` U; }}
655:       BBUS { 01 { `10ns` D/U; `300ns` U; }}
656:     } // end Waveforms
657:   } // end sequence_one
658:
659:   WaveformTable sequence_two {
660:     Waveforms {
661:       DIR { 01 { `0ns` D/U }}
662:       ABUS { LHZX { `0ns` Z; `460ns` L/H/X; `480ns` T; }}
663:       BBUS { LHZX { `0ns` Z; `460ns` L/H/X; `480ns` T; }}
664:     } // end Waveforms
665:   } // end sequence_two
666:
667:   WaveformTable sequence_three {
668:     Waveforms {
669:       BBUS { LHX { `0ns` Z; `260ns` L/H/X; `280ns` T; }}
670:     } // end Waveforms
671:   } // end sequence_three
672:
673:   WaveformTable sequence_four {
674:     Waveforms {
675:       ABUS { LHX { `0ns` Z; `460ns` L/H/X; `480ns` T; }}
676:       BBUS { 01 { `10ns` D/U; }}

```

```
677:     } // end Waveforms
678: } // end sequence_four
679:
680: WaveformTable one {
681:     Period InheritWaveformTable period_one;
682:     Waveforms {
683:         DIR { InheritWaveformTable sequence_one.DIR; }
684:         OE_ { InheritWaveformTable sequence_one.OE_; }
685:         ABUS { InheritWaveformTable sequence_one.ABUS; }
686:         BBUS { InheritWaveformTable sequence_one.BBUS; }
687:     } // end Waveforms
688: } // end one
689:
690: WaveformTable two {
691:     Period InheritWaveformTable period_one;
692:     Waveforms {
693:         DIR { InheritWaveformTable sequence_two.DIR; }
694:         OE_ { InheritWaveformTable sequence_one.OE_; }
695:         ABUS { InheritWaveformTable sequence_two.ABUS; }
696:         BBUS { InheritWaveformTable sequence_two.BBUS; }
697:     } // end Waveforms
698: } // end two
699:
700: WaveformTable three {
701:     Period InheritWaveformTable period_two;
702:     Waveforms {
703:         DIR { InheritWaveformTable sequence_one.DIR; }
704:         OE_ { InheritWaveformTable sequence_one.OE_; }
705:         ABUS { InheritWaveformTable sequence_two.ABUS; }
706:         BBUS { InheritWaveformTable sequence_three.BBUS; }
707:     } // end Waveforms
708: } // end three
709:
710: WaveformTable four {
711:     Period InheritWaveformTable period_three;
712:     Waveforms {
713:         DIR { InheritWaveformTable sequence_two.DIR; }
714:         OE_ { InheritWaveformTable sequence_one.OE_; }
715:         ABUS { InheritWaveformTable sequence_one.ABUS; }
716:         BBUS { InheritWaveformTable sequence_four.BBUS; }
717:     } // end Waveforms
718: } // end four
719:
720: WaveformTable five {
721:     Period InheritWaveformTable period_one;
722:     Waveforms {
723:         DIR { InheritWaveformTable sequence_one.DIR; }
724:         OE_ { InheritWaveformTable sequence_one.OE_; }
725:         ABUS { InheritWaveformTable sequence_two.ABUS; }
726:         BBUS { InheritWaveformTable sequence_three.BBUS; }
727:     } // end Waveforms
728: } // end five
729: } // end Timing basic
```





