Title: STIL 1450.1 Internal Resolution of 1450.1-D15

History:

- 09/21/02 - issues carried over from D14
- 10/07/02 - updates from ITC-2002 working group meeting
- 11/07/02 - updates from 11/7 phone meeting
- 11/21/02 - updates from 11/21 phone meeting
- 01/08/03 - updates from wg review of doc
- 03/27/03 - updates from 3/27 phone meeting

The working document “P1450.1 Draft 14 is now undergoing review and update.
The following are the individuals participating in the review:

1. (BC) bill_chown@ims.com
2. (BR) bennyr@taux01.nsc.com
3. (CW) cww@ee.nthu.edu.tw
4. (DD) dave_dowding@agilent.com
5. (DK) david_kellerman@teseda.com
6. (DM) denism@synopsys.com
7. (DO) don.organ@inovys.com
8. (DS) dsprague@btv.ibm.com
9. (GM) gmaston@qwest.net
10. (GR) gordon_robinson@3mts.com
11. (GW) gwilder@dal.asp.ti.com
12. (HR) hira_ranga@3mts.com
13. (IS) ishikawa@eeclkg.eec.toshiba.co.jp
14. (JD) jason_doege@inovys.com
15. (JO) jim_oreilly@agilent.com
16. (JT) jim@galois.com
17. (KD) klaus-dieter_hilliges@agilent.com
18. (LM) larry.moran@teradyne.com
19. (PW) wohl@synopsys.com
20. (RK) rkapur@synopsys.com
21. (S3) source3@calweb.com
22. (TT) tonyt@synopsys.com
23. (TW) tww@synopsys.com
24. (WG) working group

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution
DM-3	Editorial - 12.3, Pattern Tiling and use of Wait/Extend 1. A better explanation of the application rules for the Wait and Extend statements is needed. Probably should move Annex K into this section as way of explanation. 2. The syntax in Annex K differs from clause 12.3. In clause 12.3 the Wait appears on the pattern. In Annex K it is after the PatternBurst. Which way should it be? 3. Suggest the following semantic for Wait: “If there is no Wait statement following a ParallelPatternBurst, then all patterns or bursts shall complete before continuing. If there is a Wait following a ParallelPatternBurst, then processing shall continue as soon as all the patterns and bursts listed in the Wait block have completed.”	4/25 - 1. wg agreed that Annex K should be moved to the syntax example section. 2. wg tended towards the form where the Wait and Extend are attributes of the pattern, rather than free-floating statements. However, wg decided to wait for more input from Jason on concurrent pattern execution. 7/16 - sec 12.4 updated and Annex K removed by tt. 7/18 - wg would like feedback from JD wrt his proposal on pattern merging. 11/21/02 - JD agreed to review and provide input. 3/27/03 - Jason agrees that the syntax, as is, is OK.
DM-4	4/22/02 - Technical - 16.1 The ScanStructures statement should be used as a domain definition that is used to specify the environment for a pattern. As such it should not be allowed as a pattern statement.	4/25 - Peter is the primary proponent of this syntax. He will discuss this with Denis during the week of 5/6 when he is in sunnyvale. 8/30 - An alternate proposal has been made to create a new ScanChains (plural) command. This would have the semantic that all existing chains go away and only the ones in the ScanChains statement are in force. 10/7/02 - The new ScanStructures->ScanChainGroups and Pattern->ActiveScanChain statements have been added to the draft. 11/7/02 - Added example of usage of these new statements (clause 15.5)
DO-1	9/3/02 - Technical - Various comments re: variable syntax. See appendix 2 (end of this document).	10/7/02 - Issues were reviewed at wg meeting at ITC-02. Several syntax changes were made per Don’s suggestions.

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution
DS-1	Editorial - Pg 11, Section 6.1, last paragraph. This paragraph is very confusing and tough to get a grip on. We believe it is because of the over usage of the word "enumeration" within that paragraph. For example, the sentence starting with "SignalVariableEnum and IntegerEnum variables shall..." is a real mouthful.	1/3/03 - Paragraph re-written.
DS-2	Editorial - Pg 12, Section 6.6 A general statement about the confusion of the relationship between expressions. For instance, it appears that all boolean expressions, integer expressions, and real expressions are all logical expressions. Would it be helpful to have some kind of hierarchical tree or BNF type diagram to show this relationship. If this makes sense, maybe this would be best placed up front prior to going into expressions. Like maybe as section 6.2. Just a thought to maybe help clear up the explanation of expressions.	1/3/03 - Much rework of the variables and expressions clauses to clear thing up (hopefully).
DS-3	Editorial - Pg 35 Section 16.4. We still need to address the issue of X statements within procedures and macros and how fail data using X statements can uniquely identify references back to these points in the patterns. I suggested in an email putting verbage into this section to enforce a hierarchy across X statements from the mainline patterns down through the macros/procedures. Never got any replies so not sure how people feel about this. Need to discuss this at the next meeting.	1/3/03 - Added definition of a new construct to allow hierarchical references: label:proc2:proc1. See updated doc for detail.
DS-4	Editorial - Pg 39, Section 18.1, next to last paragraph starting with "(12) Type...". It say the file type shall be "one of the specified types" but no types are specified anywhere. Also it says you can specify User followed by the type name but there is no provision for this in the syntax description on pg 38. I guess you could argue this second point is actually OK because it's covered in this definition of of file_type but it seems it would be clearer if the syntax said something like: Type (User) type_name ;	01/03/2002 - Greg has created a list of file types and included it in the "recommended usage document" on the STIL web site.

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution

GM-1	Section 14 top of pg 31, the syntax:
	
	(ScanCells {
	  (< cellname_list
	   | cellname_list {
	
	  is wrong, because it implies the interpretation of the all cells before the {} are part of the {}, e.g.:
	
	    ScanCells { a b c { CellIn hoozy; }} //hoozy is on cell ’c’ only and not ’a’ and ’b’
	
	It’s also the wrong syntax for the block construct. We have two forms of ScanCell statements. The first is the single-statement form defined in dot-zero, seen in the first example in 14.2 pg 33, and it uses cellname list:
	    ScanCells a b c;
	
	The second form is seen in the example in 14.3, which is the block-statement form. This form actually DOES REQUIRE STATEMENT DELIMI ERS BETWEEN EACH CELL
	NAME (as shown in that example):
	    ScanCells { a; b; c{} }
	
	This form does NOT use the cellname-list (as currently defined).
	THEREFORE, I request the syntax description above change to:
	
	(ScanCells {
	     (cellname-ref ;)*
	     (cellname-ref { ... })*
	  })
	
	We might need to define cellname-ref, possibly instead of cellname-list, to accomplish this.
		5/23/02 - PW to review and comment on GM-1, GM-2, GM-5.
		9/21/01 - This still needs to be resolved. The annex uses braces with ; separatores, whereas the cellname_list in clause 5 has no braces and space separators (tt)
		10/7/02 - This issue was discussed and resolved. The new block syntax uses only semi-colon (or braced block) separated scan cells; whereas for compatibility with STIL0, space separated lists may be used when the statement form of ScanCells is used.

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution
GM-2	Section 14 Definition of STATE-ELEMENT. It has been requested that each STATE-ELEMENT item be defined in the same namespace as the cellnames. The current definition states "State elements are internal design nets of CELLNAME." The problem with this is that in my experience, shadow registers often appear at the same design-level as the scan register itself, and are NOT internal elements of that cell, but I would like to tag the presence of shadow registers with CellIn and CellOut. I think the current definition is too limiting, and I propose changing the quoted sentence above to: "State element names are placed in the same namespace of all scan cells." This does not disallow state-elements to be internal elements of cellname, but requires that the elements be 'uniquified', most commonly by placing the cellname as part of the state-element name, e.g., ScanCells { "a/b" { CellIn "a/b/c"; }}	5/23/02 - PW to review and comment on GM-1, GM-2, GM-5. 10/7/02 - see final resolution in GM-1
GM-3	section (6), the If (boolean_expr) definition. The sentence "The value of boolean_expr is evaluated during pattern operation" has been questioned, as being too limiting. This is one context where the value may be evaluated, but I don't think it's necessary to limit it to this context... so I propose: "The value of boolean_expr is evaluated as necessary by the application, for example it may be evaluated during sequencing through Vectors during Pattern operation, to establish a value if necessary per Vector".	1/3/03 - Suggested wording change made in the document.

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution

GM-5	This is the most complicated one (maybe?). A single cellname-ref may identify multiple elements, e.g., "mbr[0..7]". In support of the fact that we allow this construct now, I request a uniform extension of the STATE-ELEMENT references in both CellIn and CellOut statements to both support STATE-ELEMENT-LISTS, with the following additional elaboration:
	
	If the cellname-ref is a single item, then each CellOut statement present shall specify only a single STATE-ELEMENT (the STATE-ELEMENT-LIST shall contain one reference). If the cellname-ref contains a reference to multiple elements, then there shall be a one-to-one correspondence in declaration order, of STATE-ELEMENTS in all CellIn and CellOut statements to individual cellnames referenced.
	
	I don't know if we need an example of what this is trying to support, but here it is:
	
	`// In the scan chain definition of the ScanCells.`
	`c[0..4] {`
	`  If clk_p CellIn u[0..4];`
	`}`
	
	where the dependent state element u[0] corresponds to scan element c[0], etc...
		5/23/02 - PW to review and comment on GM-1, GM-2, GM-5.
		10/7/02 - see final resolution in GM-1

GM-6	Additional semantic definition for ParallelPatList Lockstep email 6/3/02. Specifically in support of cores or modules that have connected scan chains. << text too long for this summary doc >>	6/12/02 - Updated the document with GM's input as modified by TT and with input from RK. 8/1 JD: Jason identified two additional behaviors to support pattern merging: 'dead-cycle insertion' to wait through capture operations, and pattern reordering to maximize potential signal overlap (minimize non-overlapping patterns that need to be executed separately). Jason is concerned that the tiling/sequencing and lockstep operations ought to be merged. 10/7: Additional semantics as proposed by GM have been added in support of LockStep. See - Pattern->AllowInterleave.

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution
GM-9	Editorial - Suggest moving Variables block AFTER the modification clauses.	11/21/02 - WG agreed to sequence the clauses in usage order. This means to keep Variables after UserKeywords, and to move Environment after ScanStructures.
GR-8	Fail data is a different concept from Pattern data, and should not use the identical syntactic form. A new keyword expresses the fact that this is different information.	*5/23 - JD is setting up a task force to discuss fail feedback (GR-8, GR-21)*
GR-9	There is (to me at least) confusion about which parts of the language are declarative and which are obeying statements that can change the meaning of other expressions. My belief has always been that STIL is a purely declarative language.	11/21/02 - WG has reviewed this issue. The STIL standard (both 1450.0 and 1450.1) contain things like Signals and ScanStructures that are declarative. It also contains things like Patterns and PatternBursts that are proscriptive (i.e., sequences of events). The interpretation of each is defined in the standard. Additional definition or examples will be added as the areas of need are identified.
GR-11	Clause 12.1: "LockStep" needs to be differentiated from a single Pattern across the larger set of pins. For instance can the two Lockstep Patterns have different Loop structures provided that each Vector obeys Period rules?	11/21/02 - Much definition and elaboration has been provided on this subject. All aspects of this capability have been covered along with examples.
GR12	Clause 14.1: These structures are growing and growing. And yet they can't handle the latest types of scan structure from Mentor and IBM presented at ITC 2001 and TRP01.	11/21/02 - These definitions have been reviewed by the wg, and by representatives from IBM, Synopsys, and others. It is believed that these structures are now complete.
GR-13	"Fixed", what's the meaning if the WFC used to specify the value is one that implies changes? Even worse, what WaveformTable is used to say what the value means?	1/3/03 - The definition has been updated. Both cyclized mode (using a wfc) and static mode (using \e are defined).
GR-14	The "X" statement with Offsets and Iterations is not rich enough to express the complexity of real vector execution contexts (think of "the 15th vector after "foo" in the subroutine called from the 3rd vector after "bar" on the 99th time round that loop").	1/3/03 - Doug Sprague raised the same issue (see DS3). New syntax has been added to fully support this need.
GR-15	18.1Numbering of items (e.g. (12)Type) is out of sync with syntax annotations.	11/7/02 - Done
GR-16	18.2 CTL sneaks in again	1/3/03 - All inappropriate references to CTL have been removed.

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution
GR-17	19.1 Too specific for being called "BistStructures". Useful stuff (e.g. cell types used in CA designs) is missing. In (12) replace "offstate" by "offset",	11/7/01 - BISTStructures has been removed from this document.
GR-18	19.3 Looked for [1] and eventually found it. Get all references together at one point in the Std.	1/3/03 - problem corrected (I think).
GR-19	Annex E: Integers are not SignalGroups.	11/7/02 - Integer variables and constants are now defined in a separate block called “Variables”.
GR-21	Annex N The "tag" mechanism is fundamentally ambiguous when "vector splitting" occurs. Earlier comments show how it fails to adequately specify many contexts. In general I want to see clear syntactic entries identifying information as fail data.	*5/23/2002 - JD is setting up a task force to discuss fail feedback (GR-8, GR-21)*

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution
GR22	1) Some of the things that look to me like "operators" aren't mentioned as such in any of the tables. Specifically: [and] used in signal references. @ used in event time expressions , used in function parameter lists e.g. max(3,5). . used in hierachy navigation 2) The precedence of the modulus (%) operator is very unconventional. Almost all languages give it the same precedence as multiply and divide, but P1450.1 has it separated from those by + and -. 3) P1450.1/.6 make some strong distinctions between the various "types" of expression, and what operators are allowed in each type. But there seem to be many situations where there is no syntactic clue as to which of the expressions is allowed (e.g. in the examples in P1450.1, we see "sigref" expressions and "logical" expressions in exactly the same context. 4) This starts to get nasty when we add the lexical ambiguities that STIL has (e.g. is H1 an identifier or a sequence of WFCs). It's plausible to argue that we should use operand type information from earlier in the expression, or symbol table info, to guide this, but that means that some operator could only be used "one way round". For example, is fred == hhll OK when fred's a signalgroup, but not when its hhll as the signalgroup name and fred as the "unusual" sequence of WFCs. 5) I've also got some lurking suspicions that the meaning of some of these might change (in obfuscated STIL contest situations) if a Pattern is used in contexts with different sets of group names, macro names etc. active. 6) Then of course we've got an old issue of mine that I believe that the 'ticks' around the expressions shouldn't be necessary.	1/3/03 - Much work has been done on improving the syntax and description of variables and expressions.

Table 1: Summary of Issues with Draft P1450.1-D15

Ident	Issue	Resolution
GR-23	Technical - Need all keywords be reserved? See Appendix 1 (end of this doc) for elaboration on this issue.	11/21/02: Each dotted standard shall contain a table of reserved words defined in that standard (at least to the abilities of the participants and reviewers of these standards to influence the constructs present in each dotted standard). Those words are the words that are reserved if that dotted standard is invoked by the STIL { ... } statement. That table shall contain at least all single-valued words that appear as the first word in all statements defined in that standard (statements that start with words defined from other blocks, for example signal-references, are not contained in this table). IT MAY CONTAIN ADDITIONAL WORDS USED IN THAT STANDARD AT THE DISCRETION OF THE STANDARD DEVELOPERS.
RK-1	Editorial - 22.6 I do not see any place in 1450.1 where you upgrade Clause STIL0-22.6 to allow an integer_expr in the place of loopcnt. (label:) Loop integer_expr { (pattern-statements)*}	The example in STIL1-12.4 shows an example of using a ‘count’ expression. In addition, a new sub-clause STIL1-16.7 is added to explicitely allow define such.
RK-2	Editorial - Annex F This example of tied scan chains running in lock-step can be simplified and improved.	1/8/03 - This example reworked by tonyt to use the AllowShared facility as defined under LockStep.
RK-3	Technical - The CTL working group has identified the need for a scan-cell-groups statement. This is an arbitrary grouping of scan cell names that may come from multiple scan chains. CTL would like to define groups of cells that have common attributes. See Table 3 wrt name spaces.	1/3/03 - ScanCellGroups block has been added.
TT-2	Technical - BistStructures to be removed from the document. Can be handled as user-keyword in tools that need it.	8/1 - Decision to remove BISTStructures from dot1.
WG-1	Editorial - general Need to add references in the body of the document to the appropriate Annexes.	Done
WG-2	Need an explanation of \e	1/3/03 - Syntax definition and example added to the doc

Appendix 1 - Reserved words - Gordon Robinson 7/25/02

From: Gordon Robinson <Gordon_Robinson@3mts.com>
To: "Tony Taylor (E-mail)" <Tony.Taylor@synopsys.COM>,
    "Greg Maston (E-mail)" <g.a.maston@ieee.org>
Subject: Need all keywords be reserved?
Date: Thu, 25 Jul 2002 07:42:23 -0700
X-Mailer: Internet Mail Service (5.5.2653.19)

Looking ahead to the set of dots under development, I'm getting
concerned that our set of reserved keywords will grow too much.

As an example we can see many "enumerations" in CTL, each
of which has many possible values.

The STIL style up until now has been to make all of these keywords
reserved, and so make them unavailable to the user. Worse, every time
we add such words we can invalidate things the user already has. That
is why many programming languages have been reluctant to add further
reserved words, and some have even reserved from first release a number of
words they anticipate possibly using in later versions.

Many (most) of the keywords in STIL are used in constrained contexts,
and so need not be reserved for any language design and parse reasons.
For instance the values of enumerations could be treated syntactically
as Identifiers, with the semantic rule that the value is one of a predefined
set.
If, however, the language definition states that they are "reserved",
implementations
are obliged to treat their use as an error, even if they use the "just an
identifier"
mechanism to implement the enumerations.

Back in the HILO and HITEST days we went to great lengths to avoid
all reserved words. Even strong structural words like WHEN and RESET
could be used as signal names (I leave you to guess what sort of monstrosity
examples we wrote to check those pieces out).

STIL seems strange because there's this great list of reserved

words, but any of them can actually occur as a non-reserved token in vector
data,
so they're not as reserved as they might appear.

I'd like to suggest that at the very least we make enumeration values not be
reserved
in extensions to STIL, and sound the community out about whether we should
"unreserve"
some of the others.

Gordon

==

Appendix 2 - Don Organ At 03:21 PM 9/3/02

Tony,
I'm working with the 1450.4 multisite sub-group regarding multisite issues related to your 1450.1 (and 1450.3) specs.

However, in glancing over the 1450.1, the expression stuff caught my eye. As you recall, I implemented a rather extensive system in enVision that dealt with various "types" of expressions, and I have more experience with this type of stuff since then - both from the computer science theory as well as practical implementation issues. Although I haven't read the 1450.1 spec closely enough to understand it all, here are some comments:

 * 6.4 -c "suffixes" The governing entity for all this stuff seems to be the SI units - which is very extensive (<http://www.bipm.fr/>http://www.bipm.fr/). In the SI units, what you call "suffixes" here are called "prefixes" in SI - presumably since they are used before the units. I

* 6.4-c "suffixes" I think it is a bad idea to allow the "suffixes" without an associated unit. The SI specifically precludes it (#4 in section 5.4 of their brochure - <http://www.bipm.fr/pdf/si-brochure.pdf>http://www.bipm.fr/pdf/si-brochure.pdf): There is considerable ambiguity about with 10k means 1000 or 1024. Regardless of if or how you define that, people will use it differently. I think it is easy enough, and totaly unambiguous to have people use scientific notation - i.e 1.23e6. [Also, this avoids the somewhat obscure issue of using multiple "suffixes": 2kM (which can be generated if people think it is possible to multiple a number by a million simply by appending an 'M' - not something people would do, but something a program generator could potentially do).]

Also, this inconsistent with 1450-1999 section 6.13 which says "An engineering prefix fromTable 4, when present, shall be used in conjunction wiath an SI Unit fromTable 3."

* 6.4 - You don’t allow exponential notation. 1.23e6 is an integer - and I see nothing wrong with treating it as one.

* I think the tendency to defining different types of expressions (boolean_expr, integer_expr, logical_expr. real_expr, sigvar_expr, in addition to 1450-1999’s time_expr, 1450.2’s cd_expr) is the wrong approach. I think there should be a general concept of an expression, and that an expression is made up of operands and operators, and there is typing of each term (and of the result of each operator) and that the final result has a type. It is this final type that should be identified in different places within STIL. For example, the only difference between a time_expr and a dc_expr should be that the time_expr resolves to units of ’s’ (seconds), while the dc_expr results to units of ’V’ (or amps or ...). If you think there is merit in this type of unification, I’d being willing to help draft up such a description.

* 6.2 Boolean Expression - I think a boolean expression should result in either false or true. I think the concept of "0 if the result if False" is confusing and perhaps even self-contradictory. Booleans aren’t numbers, and there is really no reason that they need to be treated as numbers. People don’t need to do things like ("1+True") or ("2V * False") and I don’t think the language needs to support that.
Why "interpret the value 0 and negative values as False"? When is that useful?
If somebody wants to convert a boolean value to a numerical value, you have already given them an easy way: ’a>2 ? 1.0 : 0.0’ - the boolean sub-expression (a>2) is used in the conditional operator to choose between different values - one to use when true, the other for false.

* I do think a bitwise expression is separate from a boolean expression and could/should be a unique type.

* I don’t understand what a logical expression is. Part of it seems to be the support of the ’=’ for assignment. In C, the assignment operator is an operator and doesn’t require a different type. For example, it is legal (but unusual) in C to do the following:
int a, b;
a = (b = 3) +4; // assign 3 to b, and 7 to a.
So, in most ways, the assignment operator can be considered just another operator (it takes a left operand and a right operand, and produces a result), but with the extensions that the left operator must be an l-value (assignable).

* Typo in 3rd paragraph of 6.7. ’f’ must be uppercase ’F’ for farads in SI units. similarlly, ’w’ must be ’W’ in the Wattage assignment in the Spec example further below. (Most of the SI units we use are capitalized - except for ’s’ for seconds (’S’=Siemens)).

* Table 4 in 6.7:
Typo in first syntax example: ’23.0ns/2+16.5e-9-t2’ The 23ns/2 resolves to seconds, it is nonsensical to add something else (16.5e-9 - which has no units) to seconds.

* I don’t understand the distinction between engr_expr and real_expr - nor the need for such a distinction.

* As I understand it, a constant is just a "variable" that doesn’t change (across min/typ/max or across Categories). If so, then why is it restricted to be an integer? Why couldn’t I have a constant voltage or time?

Oops. Maybe I got carried away. Back to my day job...

-DVO-