10. Variables Block

This clause defines variables and constants. If the block is unnamed then all definitions shall be globally available to all other blocks in the STIL file. If the block has a name, then that name must be referenced by the Pattern or PatternBurst for the variables to be available.

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

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

10.1 Variables Block Syntax

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

} *// end Variables*

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

(2) **IntegerConstant**: This statement allows the definition of an named integer that has a constant value. The named constant shall be used anywhere that an integer value is allowed. An integer constant can be more widely used than an integer variable - for example if K is a constant, then it may be used in defining an indexed list of signals (SIGS[1..K] In;), as a length indicator (ScanIn K; DataBitCount K; ScanOutLength K;), as a loop counter (Loop K {} MatchLoop K {}).

(3) **Integer**: This statement or block defines an integer variable. An integer variable shall be used only in integer expressions. If the integer variable is specifed as a block then the following optional attributes may be specified:

a) **Usage Test**: This statement specifies that the variable is to be used in the translation, load, or execution of the STIL file for an ATE system. Variables without this statement may safely be ignored for the purposes of pattern exection. The default is YES - allow usage test.

b) **InitialValue**: This attribute allows the specification of the initial value that is to be assigned to the variable at the onset of each execution of the first or highest-level PatternBurst that uses this Variable. The value of the expression defined in the InitialValue statement is not established until a context where this variable is applied is initiated. This occurs under a PatternBurst that references a Variables block that contains this variable. The default initial value is zero.

c) **Values**: This statement shall specify the allowed range of values that may be assigned to the variable. The

default is no restriction on the allowed range of values.

(4) **SignalVariable**: This statement or block defines a variable that is to be used wherever a signal or group may be used. SignalVariables may be declared as either a single entity, or consist of a bracketed declaration. It is an error to access a bracketed index that is outside the declared range of a bracketed SignalVariable. A SignalVariable consumes one WFC for each index when assigned a wfc-list. When declared as a bracketed SignalVariable, a reference to that SignalVariable without brackets is equivalent to a reference of that SignalVariable across the declared bracketed range of that SignalVariable. If the signal variable is specifed as a block then the following optional attributes may be specified:

d) **Base** <**Hex | Dec**> WFC_LIST : This attribute defines the mapping from WFC characters to/from decimal or hex integer values, as defined for the Base option of SignalGroups in IEEE Std. 1450-1999 Clause 14. If no base is specified, the default is to waveform characters.

e) **Alignment**: This attribute is used to specify the mapping from integer to SignalVariable is to start with the MSB or the LSB. The default is MSB to accommodate the convention used for scan data.

f) **InitialValue**: This attribute allows the specification of the initial value that is to be assigned to the variable at the onset of each execution of the first or highest-level PatternBurst that uses this Variable. Variables declared with no InitialValue shall contain an undefined value until they are assigned a value (written to). The value of the expression defined on the InitialValue statement is not established until a context where this variable is applied is initiated. This occurs under a PatternBurst that references the Variables block that contains this variable, in combination with the PatternExec that invoked this PatternBurst.

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

10.2 Variables Example

```
1: STIL 1.0 { Design D15; }
2:    Header {
3:      Source "P1450.1 Working-Draft 16, Apr 24, 2003";
4:      Ann {* clause 11.2 *}
5:    }
6: Variables {
7:      IntegerConstant BUSTOP = 15;
8:      Integer H1;
9:      Integer H2 { Usage Test; InitialValue 13; }
10:     Integer H3 { Values 2 4 6 8; }
11:     Integer H4 { Values 1..100; }
12:     IntegerConstant RED=0;
13:     IntegerConstant GREEN=1;
14:     IntegerConstant BLUE=2;
15:     Integer COLORS { Values RED GREEN BLUE; }
16:     SignalVariable VX[1..5]{ InitialValue 11000; }
17:     SignalVariable VY[7..1]{ Base Hex AB; InitialValue FE; }
18:     WFCConstant RESET=00;
19:     WFCConstant RUN=00;
20:     WFCConstant EXTEST=00;
21:     WFCConstant INTEST=00;
22:     SignalVariable VZ[0..1] { Values RESET RUN EXTEST INTEST; }
23:     SignalVariable FOO[1..6];
24: }
25: Signals {
26:     bus1[ BUSTOP .. 0 ] Inout;
27:     SIG1 In;
```

```
28:     SIG[2..6] In;
29:   }
30: SignalGroups {
31:   GRP = 'SIG[2..6]';
32: }
33: Procedures {
34:   PROC {
35:     C { FOO=#; }
36:     V { SIG1='FOO[1]'; GRP='FOO[2..6]'; }
37:   }
38: }
39: Pattern P {
40:   Call PROC { FOO=111000; }
41: }
```

6.5 Logical Expressions (*logical_expr*)

Logical expressions define a sequence of operations to be performed. The left hand side of an expression shall be a variable and determines the type of the expression to be allowed on the right hand side (i.e., . The right hand side of the expression shall be either a single token, or an expression enclosed in single quotes.

The operator set and expression constructs supported by logical expressions is the same as defined for IEEE Std. 1450-1999, Subclause 6.13 (Timing Expressions), with additional bitwise and boolean operators supported. All logical expression operators are defined in Table 5 on page 20.

Logical expressions may evaluate to integer, real number, bitwise logical, or boolean values. The result of the last operation executed in evaluating a logical expression is the result of a logical expression, for example the result of assigning a value to a logical variable is the value of that assignment.

Logical expressions consist of:

a) References to Variables and Constants that are defined under Variables blocks that are currently applied in PatternBurst contexts.

b) References to Signals, SignalGroups, or Spec Variables that are currently defined in the PatternBurst and PatternExec context.

c) Integers, or real numbers.

d) Expressions as described in IEEE Std. 1450-1999 clause 6.13 but ***not*** including event labels nor time marks, and supporting the set of operators defined in table 5.

Consider the following examples of logical expressions:

```
42: STIL 1.0 { Design D15; }
43: Header {
44:   Source "P1450.1 Working-Draft 16, Apr 24, 2003";
45:   Ann {* clause 6.5 *}
46: }
47: Signals { S[1..4] In; }
48: SignalGroups { SIGGRP = 'S[1..4]'; }
49: Variables {
50:   IntegerConstant RUN=0;
51:   IntegerConstant STOP=1;
52:   IntegerConstant LOAD=2;
53:   IntegerConstant UNLOAD=3;
54:   Integer CMDS { Values RUN STOP LOAD UNLOAD; }
55:   Integer A;
56:   Integer B {InitialValue 1000;}
```

```
57:   Integer C;
58:   Integer E;
59:   SignalVariable W[1..4];
60:   SignalVariable VAR1;
61:   WFCConstant STOP=00;
62:   WFCConstant GO=01;
63:   WFCConstant RESET=10;
64:   SignalVariable VAR2[1..2];
65: }
66: Spec {
67:   Category CAT {
68:     R = '25ns';
69:   }
70: }
71: Pattern PAT {
72:   If 'A == 0' {}                  // False (in a boolean context)
73:   If 'A == 1' {}                  // True (in a boolean context)
74:   If 'A == 956' {}                // True (if present in a boolean context)
75:   If A {}                         // True if a > 0
76:   If 'R >= 20ns' {}               // a real number in engr units
77:   If 'A == 5' {}                  // test for a equal to 5; returns 1 if True and 0 if False
78:   If 'A <= B' {}                  // test for a less than or equal to b
79:   If 'A < min (B ,C)' {}          // use of the min function
80:   C { A = 5; }                    // set variable a equal to 5
81:   C { E = RUN; }                  // set using constant definition
82:   C { W = LH01; }                 // set signal variable to the string LH01
83:   C { B = \h FF; }                // set integer to 255
84:
85:   // Variables are optionally initialized when declared,
86:   // and are manipulated in logical expressions,
87:   // which appear in cyclized pattern statements as defined in clause 18.1.
88:   // Logical expressions may be used in the following contexts:
89:
90:   // As a boolean expression in a Pattern or PatternBurst:
91:   If 'E == LOAD' { V {SIG=1;}}
92:
93:   // In an evaluation statement in a Pattern or a PatternBurst:
94:   C { A = 'A+1'; }
95:
96:   // Within a Vector statement:
97:   C { W = ABBA;}
98:   V { SIGGRP = 'W';}
99:
100: // The following illustrates the compare of a signal variable to an variable/constant or a wfc
101:  If 'VAR1==\V STOP' {}           // compare with a var or constant
102:  If 'VAR2==HH' {}                // compae with a wfc list
      Question: How to indicate use of var or const as opposed to a wfc-list? Use \V ?
103:
104: } // end Pattern
105:
```