

## 6. Expressions

This clause defines extensions to STIL.0 Clause 6

STIL.0 defines a limited usage of expressions - see Timing Expressions using Spec Variables (STIL.0, clause 6.13), and Signal Expressions (STIL.0, clause 6.14). This standard extends these expression capabilities with additional variable types (SignalVariable, Integer, IntegerConstant, WFCConstant) and additional expression constructs. The use of operators and the expression constructs as defined for STIL.0 is unchanged. The full set of STIL.0 and STIL.1 expression operations are defined in Table 5. The detail of syntax and usage of the new expressions are defined in the following sub clauses.

### 6.1 Constant and variable expressions

Expression constructs (which are defined in the subclasses that follow) can be divided into two classes - those that can be completely resolved at parse time (i.e., constant expressions), and those that can only be resolved at run time (i.e., variable expressions).

"Constant expressions" are those that contain only literal values (e.g., 5, 10ns), or named constants (Spec-Variable, IntegerConstant, WFCConstant). As in STIL.0, statements that contain constant expressions can be fully resolved once the STIL file/stream has been fully parsed and the domain references have been resolved. Constant expressions can be used anywhere that the syntax definitions allow a literal value. The use of constant expressions provides for: 1) improved readability, 2) parameterized STIL data, 3) enhanced re-use, and 4) improved maintainability.

"Variable expressions" are those that contain named variables (SignalVariable, Integer). The use of variable expressions is intended primarily to support "design" applications. The ability of an ATE system to support variable expressions may be limited, if possible at all.

Both constant and variable expressions can be further classified into "arithmetic expressions" and "pattern data expressions". The allowed constructs of each of these expression types is defined in the following sub clauses.

### 6.2 Expression delimiters - single quotes and parenthesis

Single quotes are defined in STIL.0 as the delimiter to be used around expressions. This usage is unchanged with the introduction of new expression capabilities in STIL.1. The following rules apply to the use of single quotes:

- a) Single quotes are always the outermost symbols of an expression.

```
'T2+5ns'    // timing expression
'SIG1+SIG2+SIG3' // signal group expression
```

- b) Assignment and boolean operators shall not be used inside a single quoted expression.

```
V { 'INT := INT+5'; } // illegal
If 'INT == 5' { } // illegal
```

- c) Single quotes shall not be nested.

```
V {'BUS[1..'K+1']' = XXX; } // illegal - nesting of quotes
```

- d) Wherever expressions are allowed, a single token may be used without delimiters, unless specifically required in the syntax definition for that statement. A single token may be a literal value, a named constant, or a named variable.

```
V { INT := '5'; } // single token with quotes
V { INT := 5; } // single token without quotes
V { INT1 := INT2; } // single token without quotes
```

- e) Wherever expressions are allowed, they may be represented without delimiters, unless specifically required in the syntax definition for that statement.

```
V { INT := 'INT+1'; }
V { INT := INT+1; }
```

- f) Parenthesis may be used inside a single quoted expression.

```
V { INT := '(INT+1)*2'; }
V { '(GRP-SIG)+SIG' = 00001; } // make SIG the last wfc in the list
```

Parenthesis are the preferred method of delimiting expressions in STIL.1. Parenthesis can be used around the entire expression or inside the expression to specify the order of processing. The following rules apply to the use of parenthesis:

- a) Parenthesis may be used anywhere that single quotes are allowed in STIL.0.

```
V { (SIG1+SIG2) = XX; }
01 { (T2+5ns) D/U; }
```

- b) Assignment expressions may be constructed with or without the use of delimiters, unless specifically required by the syntax of the STIL statement containing the expression.

```
V { INT := INT+1; }
V { INT := (INT+1); }
V { (INT := INT+1); }
V { INT := (X+2)*2; } // parens required to define order
V { SIG1+SIG2 = XX; }
V { (SIG1+SIG2) = XX; }
V { (SIG1+SIG2 = XX); }
```

- c) Parenthesis are used to delimit boolean expressions. They may (or may not) be required, as dictated by the syntax of the STIL statement embodying the expression.

```
If (\W(A+B) == \wxx) { }
If \W(A+B) == \wxx { }
If (\WGRP != \w\r(K+12) X) { }
```

- d) Multiple assignment operators may be used in an arithmetic expression:

```
V { INT1 := INT2 := 0; }
V { SIG1 = SIG2 = X; } // illegal - use of multiple pat data operators
```

- e) Parenthesis may be nested:

```
V {SIG = \r(K+12) X; }
If (\WBUS[1..(K-1)] != \wXXX) { }
```

- f) Parenthesis are used to delimit parameters to functions within expressions:

```
V { INT := Min(X,Y); }
```

### 6.3 Arithmetic expressions

Arithmetic expressions are used in various STIL statements. The reference identifiers - *integer\_expr*, *real\_expr*, *boolean\_expr*, *time\_expr* - in the statement syntax definitions indicate that an arithmetic expression is expected, and also signifies the expected output from the expression evaluation. It is also allowed to use an arithmetic expression anywhere a literal arithmetic value is allowed as long as it contains only constants and literals. This subclause defines the common rules for all arithmetic expressions.

Arithmetic expressions use the following operators:

- `:=` assignment operator
- `?:` conditional assignment operator
- `==` compare for equality; return integer 1 if equal; else return 0
- `<>` compare for inequality; return integer 1 if not equal; else return 0
- `<` compare less than; return integer 1 if less; else return 0

- f) > compare greater than; return integer 1 if greater; else return 0
- g) <= compare less than or equal; return integer 1 if less or equal; else return 0
- h) >= compare greater than or equal; return integer 1 if greater or equal; else return 0

Arithmetic expressions use the following operands:

- a) Integer, IntegerConstant that are in scope according to the Variables context
- b) Spec variables that are in scope according to the Category selection
- c) Timing event labels when used inside a Timing block as defined in STIL.0
- d) Literal integers, reals, and engineering numbers

Pairs of single quotes are used as delimiters in arithmetic expressions according to the rules as defined in STIL.0. It is not allowed to nest single quotes, and the usage should be limited to simple expressions (e.g., 'T2+5ns'). Some statements require that delimiters always be used around expressions, even when the expression is a single literal value or name. If not so stated in the statement definition, then single operands may be used without any delimiters.

Paired parenthesis are the preferred method of delimiting expressions in STIL.1. Parenthesis can be used around the entire expression or inside the expression to specify the order of execution.

The following are examples of keyword identified expressions:

```
Loop 50 { }
ScanLength 100;
Loop K+5 { }
Loop 'K+5' { }
Loop (K+5) { }
```

The following are examples of assignment expressions:

```
Condition { INT := 5; }
V { INT := 5; }
C { INT := 5; }
C { INT := INT + 1; }
C { INT := K; }
C { INT := (INT==0) ? 99 : INT-1; }
```

The following are examples of boolean expressions:

```
If (INT == 6) { }
If (INT >= K*2) { }
If (INT > SPEC) { }
If (SPEC < 250ns) { }
While (INT <> 13) { }
```

The following are example of arithmetic expressions within a pattern data expression:

```
V { GRP[1..(K+1)] = \r(K+1) X; }
```

## 6.4 Pattern data expressions

Whereas arithmetic expressions operate on numeric data, pattern data expressions operate on wfc data. To differentiate the two, a different set of operators is defined:

- a) = assignment operator
- b) ?: conditional assignment operator
- c) == compare for equality; return integer 1 if equal; else return 0
- d) != compare for inequality; return integer 1 if not equal; else return 0

Pattern data expressions use the following operands:

- a) Signals
- b) SignalGroups that are in scope according to the PatternBurst context
- c) SignalVariables and WFCConstants that are in scope according to the Variables context
- d) Literal lists of wfc's

Single quotes are used as delimiters in pattern data expressions according to the rules as defined in STIL.0. It is not allowed to nest single quotes, and the usage should be limited to identifying signal groups (e.g., 'SIGA+SIGB', 'BUS[1..10]'). Some statements require that delimiters always be used around expressions, even when the expression is a single literal value or name. If not so stated in the statement definition, then single operands may be used without any delimiters.

Unlike arithmetic expressions, parenthesis are not to be used on pattern data assignment expression. Assignments should be of the form: TWOSIGS=XX; or 'SIGA+SIGB'=XX;. Parenthesis may be used when needed to control order of evaluation as in the following: '(XBUS-XBUS[5])+YBUS'=11110000; .

The following are examples of pattern data assignment expressions.

```
V { SIG = X; } // assign a wfc to a signal
V { GRP = XXXXXX; } // assign wfcs to a signal group
V { SIGVAR = XXX; } // assign wfcs to a signal variable
V { GRP[5..6] = XX; } // seective assign of wfcs
V { 'A+B+C' = XXX; } // assign wfcs to multiple signals
V { SIGVAR = \W SIG; } // assign wfc from a signal to a variable
V { SIG = \W SIGVAR[3]; } // assign wfc from a variable to a signal
V { GRP = \W HALT; } // assign a wfc-const to a signal group
V { GRP[1..12] = \W SIGVAR[1..4] XXXX \W SIGVAR[5..8]; }
```

The following are examples of pattern data boolean expressions. Note the use of \W on both sides of the expression. The \W operators is required to indicate that wfc data is being compared. The \W operator is used to extract wfc data from a named entity. The \w operator is used to indicate that literal wfc data follows. Note also, that when literal wfc data is used, it may appear on either side of the expression.

```
If (\WSIGVAR[6..5] == \wXX) { }
If (\wXX == \WSIGVAR[6..5]) { }
If (\WSIGVAR1 != \WSIGVAR2) { }
```

The following is an example of a pattern data expression inside an arithmetic expression:

```
V { INT := (\WGRP==\w010) ? 1 : 0; }
```

The following is a complete example of statements containing both arithmetic and pattern data expressions:

```
1: STIL 1.0 { Design D19; }
2: Header {
3:     Source "IEEE P1450.1/D20 - Oct 5, 2004";
4:     Ann {* subclause 6.4 *}
5: }
6: Signals { S[1..4] In; }
7: SignalGroups { SIGGRP = 'S[1..4]'; }
8: Variables {
9:     IntegerConstant RUN := 0;
10:    IntegerConstant EXIT := 1;
11:    IntegerConstant LOAD := 2;
12:    IntegerConstant UNLOAD=3;
13:    Integer CMD; // use values RUN, EXIT, LOAD, UNLOAD
14:    Integer INT;
15:    Integer HEX {InitialValue 0x1000;}
16:    Integer CC;
17:    Integer DD;
```

```

18:   SignalVariable SIGVAR1[1..4];
19:   WFCConstant STOP=00;
20:   WFCConstant GO=01;
21:   WFCConstant RESET=10;
22:   SignalVariable SIGVAR2[1..2]; // use values STOP, GO, RESET
23: }
24: Spec {
25:   Category CAT {
26:     TIME = '25ns';
27:   }
28: }
29: Pattern PAT {
30:   C {INT := 1;}
31:   If (INT) {} Else {} // Execute the If block, INT is True
32:   C { INT := 0; }
33:   If (INT) {} Else {} // Execute the Else block, INT is False
34:   If (INT == 956) {} // a boolean expression
35:   If (TIME >= 20ns) {} // a real number in engr units
36:   If (CC <= DD) {} // test for CC less than or equal to DD
37:   If (INT < min(CC,DD)) {} // use of the min function
38:   C {INT := 5;} // set variable INT equal to 5
39:   C {CMD = \W RUN;} // set using constant definition
40:   C {HEX := 0xFF;} // set integer to 255
41:   If (CMD <> LOAD) {} // integer compare to a constant
42:   C {INT := INT+1;} // integer expression
43:   C {SIGVAR1 = LH01;} // set signal variable to the string LH01
44:   V {SIGGRP = \WSIGVAR1;} // set signals equal to a variable
45:   If (\WSIGVAR2 == \WSTOP) {} // compare with a constant
46:   If (\WSIGVAR2 == HH) {} // compare with a wfc list
47: } // end Pattern

```