15. ScanStructures block

This clause defines extensions to STIL.0 clause 20

The STIL.0 syntax is extended to include additional information required for efficient simulation (i.e., eliminating the need to serially simulate load/unload cycles) of scan patterns when the design includes complex scan cells. A 'complex scan cell' is defined to be a state element that is loaded/unloaded by a single element of the scan chain data, but may contain multiple states internally. A related operation is that of loading multiple state elements from multiple elements of a scan chain. See annex L for discussion of typical applications using complex cells.

The STIL.0 syntax is also extended to support scan segments and scan groups. For more information see subclause 15.5.

All other constructs and requirements for STIL0 clause 20 are unchanged.

15.1 ScanStructures syntax

```
ScanStructures (SCAN_STRUCT_NAME) {                                          (1)
    ( InheritScanStructures SCAN_STRUCT_NAME ; )                             (2)
    ( ScanChain CHAIN_NAME {
        ScanLength integer_expr ;                                            (3)
      ( ScanOutLength integer_expr ; )                                       (4)
      ( ScanEnable logic_expr ; )                                            (5)
      ( ScanCellType (CELL_TYPE_NAME) {                                      (6)
            ( (If boolean_expr) CellIn INTERNAL-REF-LIST ; )*                (7)
            ( (If boolean_expr) CellOut INTERNAL-REF ; )*
      }
      ( ScanCells {                                                          (8)
         ( cell_ref (CELL_TYPE_NAME) ;  )*
      } )  // end ScanCells
      ( Base Hex; )  // optional for cell groups                             (9)
      ( Alignment <MSB|LSB>; )  // optional for cell groups                  (10)
    } )*  // end ScanChain
    ( ScanChainGroups {                                                      (11)
      ( GROUP_NAME { (GROUP_OR_CHAIN_NAME; )* } )*
    }) // end ScanChainGroups
}  // end ScanStructures
```

(1) **ScanStructures**: Refer to STIL.0 for the definition of the ScanStructures block and statements not defined in this extension.

(2) **InheritScanStructures** SCAN_STRUCT_NAME: This statement allows to reference another scan structure block that is to be included into the current block. All scan chains, scan cells, and scan cell types in the referenced block become part of the current block. Local definitions shall override definitions in an inherited scan structure.

(3) **ScanLength** *integer_expr* : As defined in STIL.0, with the extension: a) support for integer expressions, b) made optional, in which case the length is determined by the length of the shift data.

(4) **ScanOutLength** *integer_expr* : As defined in STIL.0, with the extension to support integer expressions.

(5) **ScanEnable** *logic_expr*: This optional statement allows designation of a single Signal or a complete expression to represent the design constraints, if any, necessary to allow access to the scan shift operations for this scan chain. See example usage in Annex C.

(6) **ScanCellType** (CELL_TYPE_NAME): In the new brace delimited form of specifying scan cells, an optional cell type name may be defined. The cell type may be unnamed, in which case it applies to all scan cells without an explicit name reference. The cell type can be named, in which it applies to all scan cells with the coresponding name. Cell types are valid only in the context of the current scan structure block or any block that inherits this block. The operation of the cell is defined by the CellIn and CellOut statements that are contained within the block. If no

Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

statements exist within the ScanCellType statement, then no scan data is to be consumed by that cell (i.e., the case of a lockup latch).

(7) (**If** *boolean_expr*): Optional conditional clause on the CellIn and CellOut statements. The value of *boolean_expr* is evaluated as necessary by the application in order to determine the appropriate activity of the scan cell. When *True*, the following CellIn or CellOut statements are applied. See subclause for information about *boolean_expr.*

INTERNAL-REF is a name and INTERNAL-REF-LIST is a list of names of internal netlist elements separated by whitespace. Internal netlist elements can be internal design nets of scan cell names (*cell_ref*). Inversion is indicated by inserting the "**!**" character before or after names. When CellIn and CellOut constructs are inherited through InheritScanStructure constructs, the names of all inherited netlist elements are prefixed with the INST_NAME (and a period) to identify specific instances of these state element names. All references to netlist elements (INTERNAL-REF, INTERNAL-REF-LIST and *cell_ref*) must be within the name space recognized by the STIL interpreter (e.g., simulator) to enable parallel simulation.

CellIn: This is an optional statement indicating that the nets in INTERNAL-REF-LIST are to be loaded with the data value corresponding to the current scan cell, with possible inversion as indicated by a "**!**" character. A "**!**" indicates inversion between the input of the scan cell and the optional name following the "**!**". If no name follows the "!" then the inversion is inside the named scan cell, between the cell input and the state element.

CellOut: This is an optional statement indicating that, when the boolean_expr is true, the INTERNAL-REF is to be unloaded (into the scan cell name) with the data value corresponding to the current scan cell, with possible inversion as indicated by a "**!**" character. A "**!**" indicates inversion between the name preceding the "**!**" and the output of the scan cell. If no name precedes the "!" then the inversion is inside the named scan cell, between the state element and the cell output.

Both **CellIn** and **CellOut** statements may be qualified by If *boolean_expr*. This indicates that the nets in the **CellIn** and **CellOut** statements are only to be considered in a pattern where the *boolean_expr* is *True*. See table 1 for valid use cases for **CellIn** and table 2 for valid use cases for **CellOut**:

Table 1—Use cases for CellIn

No CellIn statement	No information available about scan cell load operations. Full simulation is required.
CellIn statement	The parallel load operation is defined and consistent. No load operations are to be simulated.
boolean_expr CellIn statements	The parallel load operation is defined for the case when the boolean expr is true.When all boolean_expr are false the load operation must be evaluated by serial simulation. It is an error if more than one boolean expr evaluates to true.
Both simple CellIn and *boolean_expr* CellIn statements	The load operation is fully defined. If all boolean expr are false then the simple cell in block is to be executed.

Table 2—Use cases for CellOut:

No CellOut statement	No information available about scan cell unload operations. Full simulation is required.
CellOut statement	The parallel unload operation is defined and consistent. No load operations are to be simulated.

Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

boolean_expr CellOut statements	The parallel unload operation is defined for the case when the boolean expr is true.When all boolean_expr are false the unload operation must be evaluated by serial simulation. It is an error if more than one boolean expr evaluates to true.
Both simple CellOut and *boolean_expr* CellOut statements	The unload operation is fully defined. If all boolean expr are false then the simple cell in block is to be executed.

(8) **ScanCells**: This statement shall appear at most once in a scan chain block. It shall contain either: a) a space separated list of scan cell names (see *cell_ref* in subclause 6.12), or b) a brace delimited block containing a semicolon separated list of scan cell names. The new block form of this statement allows for reference to a scan type block that defines complex scan cells. See annex L for examples of this construct.

(9) **Base Hex**: This statement is optional and is used when the referenced list of scan cells is to be used as a cell group. The only attribute allowed is Hex. Note that this does not control the format of data for scan shifting which is specified in the ScanIn/ScanOut signal definition. See also the statement Alignment. For information about cell groups see subclause 15.5.

(10) **Alignment** <**MSB|LSB**>: This statement is optional and is use when the referenced list of scan cells is to be used as a cell group. This statement is only applicable when data is specified in hex and specifies whether the most significant bit of the hex data is to be aligned with the first cell of the chain (MSB) or whether the least significant bit of the hex data is to be aligned with the last cell of the chain (LSB). For information about cell groups see subclause 15.5.

(11) **ScanChainGroups**: A scan chain group is a shorthand way for specifying a set of scan chains. The names that comprise a group shall be chain names or other group names; and shall be defined either in the global ScanStructures block or the current named ScanStructures block in effect. The named groups can be referenced by the ActiveScanChains statement in a macro or procedure to specify the active scan chains for a shift operation (see subclauses 15.2 and 15.4).

Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

L.2 Example: scan cells with multiple state elements

In the following example, chain C1 is defined with 4 scan cells, cell A3 having a more complex definition. All inversions have been marked with a comment in the form /*invn*/ to facilitate explanations. Cell A3 resembles an LSSD cell, with the master shift clock being ACLK and the slave shift clock being BCLK. The other scan cells would typically have the same configuration as A3, but only A3 is detailed here for brevity.

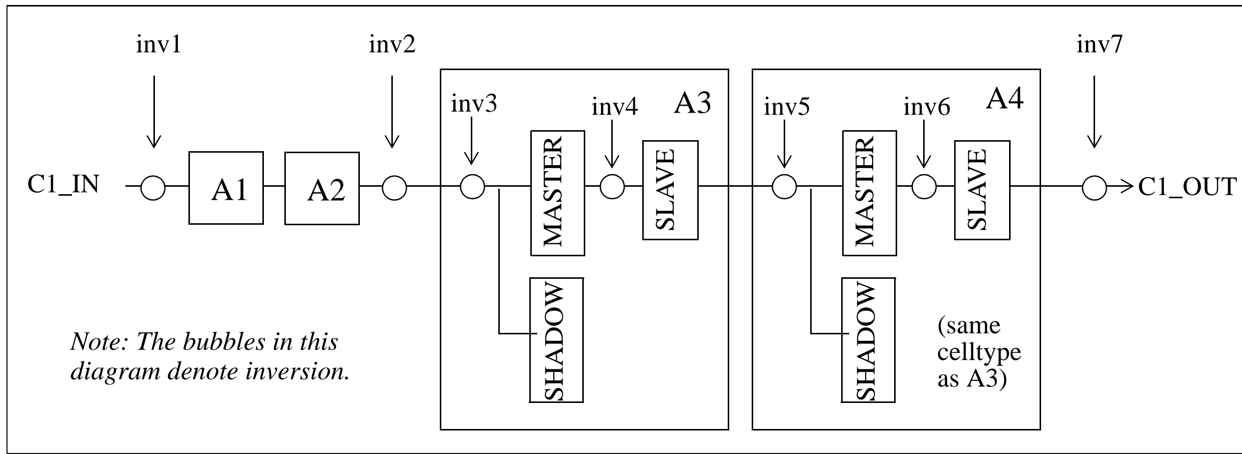

Figure 1—Example of scan chain including a cell with multiple state elements and internal inversions

This scan chain is defined as follows:

```
1: STIL 1.0 { Design D19; }
2: Header {
3:      Source "IEEE P1450.1/D20 - Aug 26, 2004";
4:      Ann {* subclause L.2 *}
5: }
6:   // The simulation-only variable SCANMODE is declared as follows:
7: Variables {
8:        Integer SCANMODE {
9:          InitialValue 2;
10:       }
11: }
12: Signals { C1_IN In; C1_OUT Out; ACLK In; BCLK In;}
13: ScanStructures G1 {
14:       ScanChain C1 {
15:         ScanIn C1_IN;
16:         ScanOut C1_OUT;
17:         ScanCellType MS {
18:             If (SCANMODE > 0)
19:               CellIn ! /*inv3*/ MASTER SHADOW ! /*inv4*/ SLAVE;
20:             if (SCANMODE :== 2)
21:               CellOut SLAVE;
22:             if (SCANMODE :== 1)
23:               CellOut MASTER ! /*inv4*/;
24:           }
25:         }
26:         ScanCells { ! /*inv1*/; A1; A2; ! /*inv2*/; A3 MS; A4 MS; ! /*inv7*/; }
27:       }
28: }
```


Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
29:
30: // The variable SCANMODE controls simulation.
31: // The load/unload procedures are setting this variable as follows:
32:     Procedures {
33:       SKEWED_LOAD {
34:         C { SCANMODE := 0; } // no parallel simulation
35:         V { C1_IN = #; ACLK = P; } // pulse A-clock
36:         C { SCANMODE := 2; } // reset
37:       }
38:       LOAD_UNLOAD {
39:         // uses current value of SCANMODE
40:         Shift { C1_IN = #; C1_OUT = #; ACLK = P; BCLK = P; }
41:         C { SCANMODE := 2; } // reset
42:       }
43:       MASTER_OBSERVE {
44:         C { SCANMODE := 1; } // MASTER_OBSERVE mode
45:         V { BCLK = P; } // pulse B-clock
46:       }
47:     }
48:
49: // The pattern block need not be concerned with the details of the scan chain and does not explicitly change the
variable SCANMODE.
50:     Pattern SCAN {
51:       "pattern 1": Call LOAD_UNLOAD { C1_IN = 0001; } // SCANMODE==2
52:       Call SKEWED_LOAD { C1_IN = 0; } // sets SCANMODE to 0, then 2
53:       V { }
54:       Call MASTER_OBSERVE; // sets SCANMODE to 1
55:       "pattern 2": Call LOAD_UNLOAD { C1_OUT = 1110; } // SCANMODE==1
56:       V { } // SCANMODE was set to 2 at the end of previous LOAD_UNLOAD
57:       "pattern 3": Call LOAD_UNLOAD { C1_OUT = 1110; } // SCANMODE==2
58:     }
```

In the pattern shown, the first vector (labeled `"pattern 1"`) calls the LOAD_UNLOAD procedure. The simulator can execute a fully parallel load of `{ C1_IN = 0001; }` because SCANMODE is 2 (from its InitialValue in the PatternBurst). The first three bits applied on c1_in are 0, the last bit is 1. This results in the following values being loaded:

— A1 is the cell closest to the scan input c1_in and is thus loaded with the last value (1) of the "0001" string, inverted as indicated by the "!" `/*inv1*/`, thus A1=0.

— A2 is the next cell, loaded with the next-to-last value (0), also inverted `/*inv1*/`, thus A2=1.

— A3 is loaded with 0 inverted twice `/*inv1*/ /*inv2*/`; within A3 the MASTER has yet another inversion `/*inv3*/`, thus A3/MASTER=1 and A3/SHADOW=0.

— A4 is the cell closest to the scan output c1_out and is thus loaded with the first value (0) inverted twice `/*inv1*/ /*inv2*/`; within A4 the MASTER has yet another inversion `/*inv5*/`, thus A4/MASTER=1 and A4/SHADOW=0.

Next, the SKEWED_LOAD procedure is called in the Pattern block. The `{ C1_IN = 0; }` this procedure must be serially simulated because SCANMODE is set to 0, and will affect the values just loaded into the scan cells. At the end, procedure SKEWED_LOAD sets SCANMODE back to 2.

The following vector is also simulated. Next, the MASTER_OBSERVE procedure is called. This procedure has no parameters, and only affects how the following call to LOAD_UNLOAD (data `{ C1_OUT = 1110; }` and labeled `"pattern 2")` is to be interpreted, by setting SCANMODE to 1:

— A4 is the cell closest to the scan output C1_OUT and is unloaded with the first value (1) of the "1110" string, inverted as indicated by the "!" `/*inv7*/`. Within A4, when SCANMODE==1, the MASTER is unloaded with

Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

yet another inversion `/*inv6*/`, thus A4/MASTER=1;.

- A3 is the next cell, unloaded with the second value (1), also inverted `/*inv7*/`. Within A3, when SCANMODE==1, the MASTER is unloaded with yet another inversion `/*inv4*/`, thus A3/MASTER=1;
- A2 is the next cell, unloaded with (1) inverted twice `/*inv2*/ /*inv7*/`, thus A2=1;
- A1 is the cell closest to the scan input C1_IN and is unloaded with the last value (0) inverted twice `/*inv2*/ /*inv7*/`, thus A1=0.

Next, LOAD_UNLOAD is again called (label `"pattern 3").` This time, SCANMODE is 2 (set at the end of the previous LOAD_UNLOAD). The same unload data { `C1_OUT = 1110;` } is now interpreted differently for cells A4 and A3:

- A4 is the cell closest to the scan output C1_OUT and is unloaded with the first value (1) of the "1110" string, inverted as indicated by the "!" `/*inv7*/`. Within A4, when SCANMODE==2, the SLAVE is unloaded, thus A4/SLAVE=0;.
- A3 is the next cell, unloaded with the second value (1), also inverted `/*inv7*/`. Within A3, when SCANMODE==2, the SLAVE is unloaded, thus A3/SLAVE=0;

Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

22. PatternFailReport

The PatternFailReport block is used to contain fail information resulting from the test of a device. As such, it is typically not part of a STIL test program file/stream, but contains references to the associated STIL test program file/stream to establish the context. Each PatternFailReport block shall contain the set of fails from the test of one device on the tester.

See clause 19 for the definition of the X (cross reference) pattern statement which is the basis for referencing fail data back to the pattern data.

22.1 PatternFailReport syntax

```
PatternFailReport (REPORT_NAME) {                                   (1)
    ( DeviceID "IDENTIFYING INFORMATION"; )                         (2)
    ( TestConditions { } )                                          (3)
    ( Pattern PAT_NAME; )                                           (4)
    ( Pattern PAT_NAME {
        ( LogStart (X_REF_ID) (CYCLE_COUNT) ; )
        ( LogStop (X_REF_ID) (CYCLE_COUNT) ; )
    } )
    ( PatternBurst PAT_BURST_NAME; )                                (5)
    ( PatternBurst PAT_BURST_NAME {
        ( LogStart (X_REF_ID) (CYCLE_COUNT) ; )
        ( LogStop (X_REF_ID) (CYCLE_COUNT) ; )
    }
    ( PatternExec PAT_EXEC_NAME; )                                  (6)
    FailData {                                                      (7)
        ( X_REF_ID ( . X_REF_ID )*                                  (8)
            SIG_NAME
            <
                | ( CYCLE_OFFSET  )* ;
                | 'observed_data' ;
                | ( CYCLE_OFFSET  'observed_data'  )* ;
            >
        )* // end fail data record
    }  // end FailData
} // end PatternFailReport
```

(1) **PatternFailReport**: The block contains fail data as produced on a tester and is associated with a STIL test program stream.

(2) **DeviceID**: This statement contains a string for the purpose of identifying the device-under-test that produced the reported results.

(3) **TestConditions** { }: This block contains statatements that identify the conditions under which the test was made. The test conditions would typically identify things such as: environmental conditions like temperature; DC set up conditions like voltage and current; timing set up conditions; ATE type and system identification. These statements shall either be Ann statements, or else user keyword statements.

(4) **Pattern**: This statement specifies the name of the pattern in the associated STIL test program file/stream that is detecting the device failures. The block form of this statement allows for the specification of start and stop points of the logging information in this fail report. The X_REF_ID parameter refers to a X statement in the pattern. The CYCLE_COUNT parameter represents the number of vector cycles (i.e., periods) that transpire to the commencement or termination of the logging function. When only the CYCLE_COUNT is present, the cycle offset is from the beginning of the pattern. If both X_REF_ID and cycle count are present the cycle offset is from the specified X_REF_ID.

(5) **PatternBurst**: This statement specifies the name of the pattern burst in the associated STIL test program file/stream that is detecting the device failures. The block form of this statement allows for the specification of start and stop points. See the definition of these parameters in the Pattern statement, above.

Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

(6) **PatternExec**: This statement specifies the name of the pattern exec in the associated STIL test program stream that is detecting the device failures.

(7) **FailData**: This begins the block containing the device failure data.

(8) **fail data record**: Each device failure record contains the fail data information as described below.

a) X_REF_ID: The first token of each record is used to identify the reference position within the pattern. It may be either a user defined name (according to the rules of STIL.0 subclause 6.8) or an integer. Reference shall always be to the last X statement encountered in the pattern execution sequence within the base pattern. If there are X references within called procedures and/or macros then the X_REF_ID is appended with dot separators to the X_REF_ID in the base pattern. For patterns with no X statements, the pattern name itself can be used as an X_REF_ID to specify the offset if from cycle 0 of the pattern.

b) SIG_NAME: This is the name of the signal that detected the failure. It may be either a name in a Signals block or a re-named single signal in a SignalGroups block. The signal name resolution follows the standard rules for domain name resolution. The signal names always refer to the names as used in the referenced pattern, however, a Signals/SignalGroups block may be included in the fail data file/stream for the purpose of defining the hex formatting of the fail data records.

c) CYCLE_OFFSET: This token indicates the cycle offset from the last the last X statement encountered in the pattern execution sequence. This is an optional field, and if not specified, a cycle offset of 0 is assumed. The cycle number for the labeled vector shall be 0. If the last last X statement encountered in the pattern execution sequence is within a Loop or Shift block, then the cycle offset is from the first occurence of the X reference in the loop or shift.

d) *observed_data*: This field is optional and specifies the observed state(s) of the failing signal. The observed data is enclosed in single quotes. In full data capture mode this is a string of observed states - with a state specified for each cycle (either H/L/T for failing cycles, or X for cycles for which no compare is done). Compaction of this observed data is supported by expressing the the data using the \h, \r, and \e constructs as defined in STIL.0 Clause 21.1.

If there is no Signals or SignalGroups blocks in the fail report file/stream then, by default, the observed_data is an event list comprised of the compare event characters L, H, X, and T (low, high, don't-care, and tri-state) as defined in STIL.0 Clause 18.2 Table 10.
If there is a Signals or SignalsGroups block in the fail report that defines the observed signal name, then the "Base Hex;" statement can be used to specify hex formatting as the default. Note that the more compact hex logging (with 4 cycles represented by each hex character) can be done if only L and H output events are possible.

22.2 PatternFailReport example

The following are examples of syntax constructs. See annex N for a complete example of pattern fail report statements.

```
59: STIL 1.0 { Design D19; }
60: Header {
61:     Source "IEEE P1450.1/D20 - Aug 26, 2004";
62:     Ann {* subclause 22.2 *}
63: }
64:
65: Signals {
66:     PO1 Out;
67:     PO2 Out;
68:     SO1 Out; SO2 Out; SO3 Out; SO4 Out;
69:     SO5 Out { Base Hex LHT; }
70: }
71:
```


Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.

```
72: PatternFailReport {
73:     Pattern PAT;
74:     PatternBurst BRST;
75:     PatternExec EXE;
76:     FailData {
77:       13 PO1;  // example with pattern-unit=13; signal=PO1
78:       PU14 PO1;  // example with a pattern-unit identifier
79:     42 SO1 19;  // example with cycle offset (i.e. scan cell in a shift)
80:       45.LU SO1 16;  // example with reference inside a macro/proc (i.e the LoadUnload)
81:       46.M1.LU S01 16;  // example of two levels of macro/proc
82:     59 PO2 23'H';  // example of fail state
83:       64 SO2 6 16 24 25 338;  // example of multiple fails (5 in this case)
84:       72 SO3 13'H' 19'L' 45'L' 63'L';  // example of multiple fails with fail states
85:       88 S04 0'HHHLLLLHHH' 56'HHHL';    // example of data capture format
86:       Pattern PO1 19023'H';  // example referencing cycle 19023 from the beginning of the pattern
87:       "ABIST-30" SO5 22 '52 \r20 00 A \e L';  //  @ cycle 22: HHLT ...80 L's... TTL
88:       }  // end FailData
89: }  // end PatternFailReport
```


Copyright © 2004 IEEE. All rights reserved.
This is an unapproved IEEE Standards Draft, subject to change.