

1 Syntax defs – composite

2
3 data-type:

4 stil-type - STIL block references:

5 **Signals**

6 **SignalGroups**

7 **PatternBurst**

8 **Timing**

9 **Category**

10 **Spec**

11 **Selector**

12 **ScanStructures**

13 **Pattern**

14 **Procedures**

15 **MacroDefs**

16 **DCLevels**

17 **DCSequence**

18 **Environment**

19 var-type – variables and expressions:

20 **Signal** (*allow: signals, groups, signal variables, sigref-expr*)

21 **Integer** (*allow: integers, integer constants, integer-expr*)

22 **SignalVariable** (*allow: wfc strings, \W, \r, \h, etc.)*)

23 **Real** (*allow: spec variables, eng-units, real-expr*)

24 **String** (*text string in double quotes*)

25
26 **BinDefs** (BIN_DEF_NAME) { *// soft bin definitions*

27 (**Bin** < **Pass** | **Fail** > SOFT_BIN_NAME (SOFT_BIN_NUMBER);)* *// Incr pass-bin if TestResult == Pass*

28 (**Category** BIN_CATEGORY_NAME {

29 (**Bin** < **Pass** | **Fail** > SOFT_BIN_NAME (SOFT_BIN_NUMBER);)*

30 }) **// End category*

31 } *// End bindefs*

32
33 *Q1: How/where is TestResult defined? Does it need to be defined in every test-name?*

34
35 **BinMap** (BIN_MAP_NAME) { *// soft to hard bin mapping*

36 (SOFT_BIN_NAME -> HARD_BIN_NUMBER;)*

37 ([BIN_CATEGORY_NAME.SOFT_BIN_NAME] -> HARD_BIN_NUMBER;)*

38 } *// end binmap*

39 *Q2: What is the allowed syntax for "bin" arithmetic? or bin functions? i.e., ((speed.mhz200 && volts.v1) ||*
40 *(speed.mhz300 && volts.v2)) && func1 ==> 1;*

41
42 **TestMethod** TEST_METHOD_NAME {

43 (**Inherit** TEST_METHOD_NAME;)

44 (< **In** | **Out** > (**Const**) (**Once**) data-type < NAME | NAME [INDEX] > (= expr);)* *// test method parameters*

45 } *// end TestMethod*

46

```

47 TestObject TEST_METHOD_NAME TEST_OBJECT_NAME {
48   ( <In | Out > (Const) (Once) data-type < NAME | NAME [INDEX] > (= expr );)* // test object parameters
49   ( PreActions {
50     (
51       ( If <boolean_expr> | Elseif | Else { } ) // optional If clause
52       ( Assign <NAME | NAME [INDEX] = expr ;)* // Or other operations such as invert
53       ( Bin = (CATEGORY.)SOFT_BIN_NAME); // Binning on entry ports is not common, but should be allowed
54       ( Bypass); // Bypass and exit this test object immediately
55   Q3: Do we need the ability to specify the return state (Pass or Fail) when bypassing?
56   i.e. (Bypass <ReturnState>); ? Or will we simply assume ReturnState == PASS when bypassing?
57     ( <
58       Exit; // exit this test_flow
59       | Stop; // stop this test_program (i.e., the current On* operation)
60     > )
61     ( } ) // (optional) end If-Elseif-Else (Conditional pre-actions)
62   )* // end of optional pre-actions list
63 } // end PreActions
64 ( PostActions {
65   (
66     ( If <boolean_expr> | Elseif | Else { } ) // optional IF clause – common post-actions will not be conditional
67     ( Assign <NAME | NAME [INDEX] = expr ;)*
68     ( Bin = (CATEGORY.)SOFT_BIN_NAME);
69     ( <
70       Exit; // exit this test_flow
71       | Stop; // stop this test_program (i.e., the current On* operation)
72     > )
73     ( } ) // end (optional) arbiter If-Elseif-Else (conditional post-actions)
74   )* // End of optional common post-actions list
75 } // End Common PostActions
76 ( Arbiter {
77   ( If <boolean_expr> | Elseif | Else {
78     ReturnState = <Pass | Fail>;
79   } )*
80 } )
81 ReturnState (<Pass | Fail>) { // TestObject has only Pass and Fail as conditions for condition-specific post-actions
82   ( PostActions {
83     ( // start of optional post-actions list
84       ( If <boolean_expr> | Elseif | Else {
85         ( Assign <NAME | NAME [INDEX] = expr ;)*
86         ( Bin = (CATEGORY.)SOFT_BIN_NAME);
87         ( <
88           Exit; // exit this test_flow
89           | Stop; // stop this test_program (i.e., the current On* operation)
90         > )
91         ( } ) // end optional If-Elseif-Else (conditional post-actions)
92       )* // End optional return-state-specific post-actions list
93     } ) // End return-state-specific PostActions

```



```

141         | Stop; // stop this test_program (i.e., the current On* operation)
142     > )
143     ( } ) // end (optional) arbiter If-Else-Else (conditional post-actions)
144 )* // End of optional common post-actions list
145 } // End Common PostActions
146 ( Arbiter {
147     ( If <boolean_expr> | Elseif | Else {
148         ExitPath = <EXIT_PORT_NAME>;
149     } ) *
150 } )
151 ( ExitPath (<EXIT_PORT_NAME>) { // port_name is required - it's referenced by entry port actions Bypass statement
152     // or by Arbiter ExitPath statement
153     ( PostActions {
154         ( // start of optional post-actions list
155         ( If <boolean_expr> | Elseif | Else {
156             ( Assign <NAME | NAME [INDEX] = expr ;)*
157             ( Bin = (CATEGORY.)SOFT_BIN_NAME);
158             ( <
159                 Exit; // exit this test_flow
160                 | Stop; // stop this test_program (i.e., the current On* operation)
161             > )
162             } ) // end optional If-Else-Else (conditional post-actions)
163         } ) // End optional port-specific post-actions list
164     } ) // End port--specific PostActions
165     Next (= TEST_NODE_NAME | Unconnected); // go to next or named node in this test_flow
166     // or to next or named node in this test_flow
167     // connect to a following node
168 } ) * // end ExitPath
169
170 } ) * // end TestNode
171 } * // End TestFlow
172 Q4: Use of modifier "Mutable, Const, Private" on variable definitions? CONST implies parameter value
173 cannot be changed during execution. MUTABLE implies that parameter value can be changed during
174 execution.
175 Q5: What is allowed syntax for an initialization expression? How to reference the current element - e.g., In
176 Integer INT[5] = '2 * #'; In Integer SIN[128] = 'sine(#)'
177 Q6: Use instance_name.var_name to access test-nodes or test-methods that are created as instances. We need a section that
178 defines allowed expression syntax.
179
180 TestProgram (TEST_PROGRAM_NAME) {
181     ( Device "DEVICE IDENTIFYING INFORMATION"; )
182     ( Position INTEGER INTEGER; ) // x/y position from top left of window
183     ( OnLoad < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
184     ( OnPatternLoad < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
185     ( OnStart < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
186     ( OnReset < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
187     ( OnPowerDown < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
188     ( OnFinish < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )

```

```

189 ( OnLotStart < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
190 ( OnLotEnd < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
191 ( OnWaferStart < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
192 ( OnWaferEnd < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
193 ( OnSiteStart < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
194 ( OnSiteEnd < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
195 ( OnMultiSiteEnable < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
196 ( OnMultiSiteDisable < TEST_FLOW_NAME | TEST_OBJECT_NAME | TEST_METHOD_NAME >; )
197 Q8: Need to define the exact semantics of each of the On* statements; possibly with a flow chart.
198 Q9: Need definition of multi-site operation.
199 ( ProgramOption <In | Out > data-type <NAME | NAME [SIZE]> (= expr) ;)* // operator information
200 ( GlobalVariable (Const) data-type <NAME | NAME [SIZE]> (= expr) ;)* // hidden from operator
201 ( BinDefs BIN_DEF_NAME; )
202 ( BinMap BIN_MAP_NAME; )
203 Q10: Do BinDefs and BinMap need to be unique per-site?
204 }

```