

IEEE P1450.4 - Flow Extension

Change history for D14:

- 5/25/05 - removed Category, Spec, Selector as block types that can be passed as params to a test method
- 5/25/05 - renamed "Complex" to "Compound"
- 5/25/05 - alphabetized the block types, var types, and On*
- 5/25/05 - added OnException as an entry point event
- 8/8/05 - changes per 8/8/05 syntax sub-group mtg
- 8/15/05 - changes per 8/15/05 syntax sub-group mtg
- 8/22/05 - changes per 8/22/05 syntax sub-group mtg
- - D14 is to be frozen and D15 will be used to address further issues
- - D14 is completed as far as structural syntax of the language

Remaining questions/issues for D14:

- 6/1/05 - binning and exit port syntax needs to be added
- 6/1/05 - use of InitialValue or := for parameter initialization
- 6/1/05 - need to reconcile with stuff in D11 and D13
- 6/1/05 - are Category/Selector scoped or sticky, and where to be allowed
- 8/8/05 - Should we have domain names for *Defs blocks, and *Defs references in TestProgram block?
- 8/8/05 - Should we allow multiple BinDefs? Inside or outside of a TestProgram block? With domain names?

1. Syntax summary for 1450.4

```

1: stil_block_type =
2:   DCLevels
3:   | DCSequence
4:   | Environment
5:   | MacroDefs
6:   | Pattern
7:   | PatternBurst
8:   | PatternExec
9:   | Procedures
10:  | ScanStructures
11:  | SignalGroups
12:  | Signals
13:  | Timing
14:
15: real_var_type =
16:  Capacitance // F (Farads)
17:  | Compound // combinations of types
18:  | Current // A (Amperes)
19:  | Double
20:  | Frequency // Hz (Herz)
21:  | Gain // db (Decibels)
22:  | Inductance // H (Henries)
23:  | Length // m (Meters)
24:  | Power // W (Watts)
25:  | Real
26:  | Resistance // Ohm (Ohms)
27:  | Temperature // Cel (Degrees Celsius)
28:  | Time // s (Seconds)
29:  | Voltage // V (Volts)
30:
31: var_type =
32:  | Boolean // True/False or Pass/Fail
33:  | Integer
34:  | SignalRef
35:  | SignalVariable
36:  | String

```

```

37: | real_var_type
38: | stil_block_type
39:
40: initial_value_stmt =
41: InitialValue
42:   ( integer_expr )+ // allow if var of type integer_expr
43:   | ( sig_ref_expr )+ // allow if var of type sig_ref_expr
44:   | ( < True | False > )+ // allow if var of type Boolean
45:   | ( string )+ // allow if var of type String
46:   | ( wfc_string )+ // allow if var of type SignalVariable
47:   | ( real_expr )+ // allow if var of type real_var_type
48:   | ( BLOCK_NAME )+ // allow if var of type stil_block_type
49: ;
50:
51: execute_stmt =
52: FUNC_NAME ; // name of - TestMethod, TestObject, or TestFlow
53: | FUNC_NAME {
54:   ( VAR_NAME = expr ; )*
55:   ( VAR_NAME = [ expr ( expr )+ ] ; )*
56: } // end execute_stmt
57:
58: exit_port_stmt =
59: ( ExitPort < Pass | Fail > ; )
60: | ( ExitPort < Pass | Fail > {
61:   < BinAndStop ;
62:   | Goto FlowNode FLOW_NAME ; // only allowed in a flow_node
63:   | GoTo Next | End > ; // only allowed in a flow_node
64:   | other? >
65: } ) // end ExitPort
66: | ( ( If bin_expr { ExitPort ... } )+
67:   ( Else { ExitPort ... } )
68: )
69: =====
70: Variables ( VAR_DOMAIN ) { // extensions to 1450.1
71:   var_type VAR_NAME ( Const ) ( Operator ) ;
72:   var_type VAR_NAME ( Const ) ( Operator ) {
73:     ( Length integer ; ) // array of var_type
74:     ( initial_value_stmt )
75:   }
76: }
77:
78: =====
79: TestMethodDefs {
80:   ( TEST_METHOD_NAME {
81:     ( Inherit TEST_METHOD_NAME ; )
82:     ( Parameter VAR_NAME ( < In | Out | InOut > ) ; )*
83:     ( Parameter VAR_NAME ( < In | Out | InOut > ) { initial_value_stmt } ) *
84:     ( PreActions { } )
85:     ( PostActions { } )
86:     ( exit_port_stmt )
87:   } ) * // end test_method
88: } // end TestMethodDefs
89:
90: =====
91: TestObjectDefs {
92:   ( TEST_OBJECT_NAME {
93:     ( Category CATEGORYNAME ; ) *
94:     ( Selector SELECTORNAME ; ) *
95:     ( Variables VAR_DOMAIN ; ) *
96:     ( Parameter VAR_NAME ( < In | Out | InOut > ) ; ) *
97:     ( Parameter VAR_NAME ( < In | Out | InOut > ) { initial_value_stmt } ) *
98:     ( PreActions { } )
99:     ( TestExec execute_stmt ) *
100:    ( PostActions { } )
101:    ( exit_port_stmt )

```

```

102:     })* // end test_object
103: } // end TestObjectDefs
104:
105: =====
106: TestFlowDefs {
107:   ( TEST_FLOW_NAME {
108:     ( Category CATEGORYNAME ; ) *
109:     ( Selector SELECTORNAME ; ) *
110:     ( Variables VAR_DOMAIN ; ) *
111:     ( Parameter VAR_NAME (< In | Out | InOut > ) ; ) *
112:     ( Parameter VAR_NAME (< In | Out | InOut > ) { initial_value_stmt } ) *
113:     ( Title STRING ; )
114:     ( FlowNode (NODE_NAME) {
115:       ( PreActions { } )
116:       ( TestExec execute_stmt ) *
117:       ( PostActions { } )
118:       (exit_port_stmt )
119:     } ) * // end FlowNode
120:   } ) * // end test_flow_name
121: } // end TestFlowDefs
122:
123: =====
124: TestProgram TEST_PROGRAM_NAME {
125:   DUTType "DUT NAME STRING";
126:   SocketDef "SOCKET NAME STRING";
127:   ( Category CATEGORYNAME ; ) *
128:   ( Selector SELECTORNAME ; ) *
129:   ( Variables VAR_DOMAIN ; ) *
130:   BinDefs < Pass | Fail > {
131:     SoftBin BIN_NAME { }
132:     HandlerMap MAP_NAME { }
133:     Axis AXIS_NAME {
134:       SoftBin BIN_NAME
135:       HardBin integer ...
136:     }
137:   } // end BinDefs
138:   EntryPoints {
139:     ( OnException execute_stmt )
140:     ( OnFinish execute_stmt )
141:     ( OnLoad execute_stmt )
142:     ( OnLotEnd execute_stmt )
143:     ( OnLotStart execute_stmt )
144:     ( OnMultiSiteDisable execute_stmt )
145:     ( OnMultiSiteEnable execute_stmt )
146:     ( OnPatternLoad execute_stmt )
147:     ( OnPowerDown execute_stmt )
148:     ( OnReset execute_stmt )
149:     ( OnSiteEnd execute_stmt )
150:     ( OnSiteStart execute_stmt )
151:     ( OnStart execute_stmt )
152:     ( OnWaferEnd execute_stmt )
153:     ( OnWaferStart execute_stmt )
154:   } // End Entry Points
155: } // end TestProgram
=====

```

2. Semantic Rules

FUNC_NAME: This is the name of either a TestMethod, TestObject, or TestFlow and is used in the *execute_stmt* that invokes the function. These names all exist in the same name space and all names must be unique across all function types. Name conflicts are error conditions.

VAR_DOMAIN: This is the name assigned to a set of variables in a Variables block. If the Variables block is unnamed, then the variables within it are globally available. A named domain Variables block is allowed to re-define a variable in the global block. However, all named domains that are referenced in a given context shall not allow name conflicts.