Summary of Syntax Definitions for STIL-Flow Extensions (P1450.4)

Table 1: History

<table>
<thead>
<tr>
<th>Date</th>
<th>By</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>12/17/98</td>
<td>tony taylor</td>
<td>Original document - created from notes and discussions at STIL extension meetings</td>
</tr>
<tr>
<td>1/14/99</td>
<td>tony taylor</td>
<td>Updated as a result of review at P1450.4 meeting on 1/11, 1/12. Many of the syntax constructs were generated by Don Organ at that meeting.</td>
</tr>
<tr>
<td>1/27/99</td>
<td>tony taylor</td>
<td>Added a new test case suggested by Ernie Wahl - using multiple categories within a test node.</td>
</tr>
<tr>
<td>2/23/99</td>
<td>tony taylor</td>
<td>Adjustments to syntax to improve clarity. Added ebnf definitions back into the document.</td>
</tr>
<tr>
<td>3/8/99</td>
<td>tony taylor</td>
<td>More tweaks to the syntax based on review with Greg Maston, and further discussion via phone conferences: make Axis available to a Test object; removed keyword ‘Eval’; added Algorithm to the Plot object; added DCLevel to the Test object; changed Category to a list in the Test object; removed the type definition from the ProgramOption statement; added type definition to Spec variable definition. Added a section defining each of the blocks of the data model.</td>
</tr>
</tbody>
</table>

The P1450.4 standard contains the definition of the data blocks necessary to specify the sequence of activities that are to be performed on each device in order to “test” that device. The definitions that are contained in this document depend on other definitions in the base IEEE-P1450 document as well as Test Method definitions that are being developed in parallel with this work.
Data Model for STIL-Flow Extensions

Flow Syntax Block Overview STIL (P1450.4)

There are several new blocks of information used in the definition of the Test Program Flow in STIL. These blocks are depicted graphically in the above data model. This section describes the main purpose of each of the blocks. The individual statement syntax for each of the blocks is defined in the section that follows.
Axis name \{ \}  - The Axis block is used to define parameters and usage for the purpose of performing a search operation. The typical use of axis objects is in the generation of a Shmoo plot.

Bin name \{ \}  - The Bin block is a node in a flow that is used as a terminal or counter, and is used to collect information about the flow. The Bin may be a “hard” bin - i.e., the termination of a path through the flow. The bin may be a “soft” bin which collects information and then continues through the flow.

BinMap name \{ \}  - The BinMap block is used to specify bin number mapping for each of the terminal bins in a given test program.

Category - A Category object is a derived object that is to be created at run time in the STIL environment. It is formed by collecting the set of variables in all Spec objects that are defined for the specified category.

DCLevels name \{ \}  - The DCLevels block specifies all of the logic levels and supply levels used by the device. This details of this block are being defined by the P1450.2 working group.

Flow \{ \}  - A flow object contains a set of nodes (Flow, Test, and Bin object) that are to be traversed in some order. The flow object is often referred to as a “sub-flow” if it is not the top-most flow in the test program, or a “composit-test” if it is to be considered as a test itself.

Node - A Node object is an abstract object that is used to define common attributes of any of the specific instances of nodes in a flow. The allowed node instances are: Flow, Test, and Bin.

PatternBurst name \{ \}  - A PatternBurst object specifies the list of patterns that are to be executed by a test program. This is defined in the base 1450 document

Plot name \{ \}  - A Plot object is the definition of the required parameters for the performing a Shmoo plot on a device.

Selector \{ \}  - A Selector object specifies the Min/Typ/Max values to be used for each of the spec-variables that are used by a test program. This is defined in the base 1450 document.

Test name \{ \}  - A Test object is a node that performs an actual test on the device. The activity to be performed and the set of required or optional data that is required for each test is dependent on the TestMethod that is referenced by the Test. The PatternExec as defined in the base 1450 is the simplest form of an execute functional test. The full set of TestMethods is being defined by the 1450.5 working group.

TestProgram name \{ \}  - The TestProgram block is the top level definition of the entry points into a flow for a given test program execution. There are typical several entry conditions that can activate the various sub-flows, such as “start running a test” or “reset a test”. This block also references variables that are used for controlling test program operation (Note: The set of program option variables would typically be displayed to a user when running the test program in non-debug mode.)
Timing name { } - A Timing object specifies the wave shape and event times to be used for each of the signals and waveform on those signals that are used by a test program. This is defined in the base 1450 document.

**Flow Syntax Extensions to STIL (P1450.4)**

Axis AXIS_NAME {
    Title "string";
    NumberSteps 'expr';
    Variable VAR_NAME {
        Start 'var_expr';
        Stop 'var_expr';
        SignalGroup 'sigref_expr';
    }
} // end of Axis

Bin BIN_NAME {
    BinType ( Pass | Fail | Status ) ;
    BinNumber 'expr';
    ( CheckOverFlow ( True | False ); )
    ( MaxBinCount 'expr'; )
    ( Exit {
        ( 'exit_port_expression'; )* 
        ( 'exit_port_expression { 'VAR_NAME = expr'; } )* 
    })* // end of Exit
} // end of Bin

BinMap BIN_MAP_NAME {
    ( Bin BIN_NAME = 'expr'; )* 
} // end of BinMap

Flow FLOW_NAME {
    LoopExpr 'expr';
    LoopNotify ( TRUE | FALSE ) ;
    ( Node NODE_NAME FLOW_TEMPLATE {
        ( Category ( CATEGORY_NAME)+ ; )
        GoTo {
            ( NODE_NAME; )+
            ( NODE_NAME { 'port_expr'; } )+
        } )* // end of Node
    } // end of Flow

Plot NAME {
    Title "string";
}
(Axis AXIS_NAME; )*
Algorithm < Fixed | Stretch | Margin >; // Fixed - all unspecified vars unchanged
// Stretch - all unspecified vars scaled
// Margin - one parameter at a time
}

// end of Plot

Test TEST_NAME {
  LoopExpr ‘expr’;
  LoopNotify (TRUE | FALSE);
  (Timing TIMING_NAME; )*
  (DCLevels LEVELS_NAME; )*
  (PatternBurst PAT_BURST_NAME; )*
  (Eval ‘expr’;)
  (Plot PLOT_NAME; )*
  (Category (CATEGORY_NAME)+; )
  (Selector SELECTOR_NAME; )*
  (Exec TEST_METHOD_NAME {
    (test_parameter_stmts; )*
  }); // end of Exec
  (Exit {
    (‘exit_port_expression’; )*
    (‘exit_port_expression { ‘VAR_NAME = expr’; } )*
  }); // end of Exit
}

// end of Test

TestProgram NAME {
  (Device “string”);
  (BinMap BIN_MAP_NAME;)
  (OnStart SUB_FLOW_NAME;)
  (OnReset SUB_FLOW_NAME;)
  (OnLoad SUB_FLOW_NAME;)
  (OnPowerDown SUB_FLOW_NAME;)
  (OnInitFlow SUB_FLOW_NAME;)
  (ProgramOption ‘OPT_NAME = expr’; )*
}

// end of TestProgram
Changes to the STIL 1450 Spec Block and Expressions

The following additions are required in the definition of the Spec variables in the basic STIL-1450 definition. Note: The initial STIL definition only supported variables of type ‘Time’, so the type was assumed to be always the same.

Definition of variable types

In the Spec block, the type of the variables may be specified immediately following the name of the variable. For compatibility with 1450 STIL, the default value is Time.

```plaintext
VAR_NAME Boolean | Integer | Real | Voltage | Current | Time = ' expr ' ;
VAR_NAME Boolean | Integer | Real | Voltage | Current | Time { }
```

New Expression Operators

The following operators need to be added to the table of allowed operators (table 5 in the STIL document).

- `&` bitwise AND operator
- `|` bitwise OR operator
- `&&` Logical AND operator
- `||` Logical OR operator
**Flow Example**

This section is an example of a test program flow to illustrate the STIL syntax described herein. The diagram below shows graphically the example flow. The flow nodes named CONT and BASIC _FUNCT are test nodes that exist within the main flow. The flow nodes named HV_FUNC, LV_FUNC, and STRESS are nodes that exist within a sub flow and hence may be re-used. The nodes named FLOW300, FLOW200, and FLOW100 are sub-flows that are defined by the Flow block named FLOWx00. The nodes named P1..P3 and F1..F7 and ‘pass’ bins and ‘fail’ bins respectively.

```
TestProgram EXAMPLE {
    DeviceType “IC-xyz”;
    OnStart MAIN_FLOW;
}
Flow MAIN_FLOW {
    Node CONT {
        GoTo { BASIC_FUNC; F1; }
    }
    Node BASIC_FUNC {
        GoTo { FLOW300; F2; }
    }
    Node FLOW300 FLOWx00 {
        Category SPEED_300MHZ;
        GoTo { P1; FLOW200.1; F3; FLOW200.2; }
    }
    Node FLOW200 FLOWx00 {
        Category SPEED_200MHZ;
        GoTo { P2; FLOW100.1; F4; FLOW100.2; }
    }
    Node FLOW100 FLOWx00 {
        Category SPEED_100MHZ;
        GoTo { P3; F5; P6; F7; }
    }
```

Flow FLOWx00 {
  Entry { HV_FUNC, Default; STRESS; }
  Node HV_FUNC { GoTo { LV_FUNC; Port 2; } }
  Node LV_FUNC { GoTo { STRESS; Port 3; } }
  Node STRESS { GoTo { Port 1; Port 4; } }
}

Test CONT {
  Selector AC_LOOSE;
  PatternBurst ABC;
  Timing DEF;
  DCLevels GHI;
  Exec Continuity { // execute the named test method
    PinsToTest ALL_PINS;
    IForce ‘-200mA’;
    ShortThreshHold ‘-200mV’;
    OpenThreshHold ‘-1.5V’;
  }
  Exit {
    ‘TestResult == Pass’; // exit port 1, if true
    ‘TestResult == Fail’; // exit port 2, if true
  }
}

Test BASIC_FUNC {
  Exec FunctionalTest { }
  Exit {
    ‘TestResult == Pass’; // exit port 1, if true
    ‘TestResult == Fail’; // exit port 2, if true
  }
}

Test HV_FUNC {
  Exec FunctionalTest { }
  Exit {
    ‘TestResult == Pass’; // exit port 1, if true
    ‘TestResult == Fail’; // exit port 2, if true
  }
}

Test LV_FUNC {
  Exec FunctionalTest { }
  Exit {
    ‘TestResult == Pass’; // exit port 1, if true
    ‘TestResult == Fail’; // exit port 2, if true
  }
}

Test STRESS {
  Exec FunctionalTest { }
  Exit {
    ‘TestResult == Pass’; // exit port 1, if true
    ‘TestResult == Fail’; // exit port 2, if true
  }
}
Flow Example #2

This example shows a flow wherein a test node (HV_FUNC) is passed in 3 different category names and has the responsibility of testing each device at each category until a Pass condition occurs, or until all categories Fail. The result as to which category passed is used as the binning criteria at the end of the flow.

```
TestProgram EXAMPLE2 {
    DeviceType "IC-xyz";
    OnStart MAIN_FLOW;
}
Flow MAIN_FLOW {
    Node CONT {GoTo { BASIC_FUNC; F1; }}
    Node BASIC_FUNC {GoTo { HV_FUNC; F2; }}
    Node HV_FUNC {
        Category SPEED300MHZ, SPEED200MHZ, SPEED100MHZ;
        GoTo { LV_FUNC; F3; } 
    }
    Node LV_FUNC { GoTo { STRESS; F4; }}
    Node STRESS { GoTo { P1; F5; }}
    Node SWITCH { GoTo { P1; P2; P3; }}
}
Test CONT {
    Exec Continuity { }
    Exit {'TestResult == Pass';} // exit port 1, if true
    Exit {'TestResult == Fail';} // exit port 2, if true
}
Test BASIC_FUNC {
    Exec FunctionalTest { }
    Exit {'TestResult == Pass';} // exit port 1, if true
```
Test HV_FUNC {
Selector AC_LOOSE;
PatternBurst ABC;
Timing DEF;
DCLevels GHI;
Exec FunctionalTest { // execute the named test method
  PinsToTest ALL_PINS;
  IForce ‘-200mA’;
  ShortThreshHold ‘-200mV’;
  OpenThreshHold ‘-1.5V’;
  Category = 1;
}
Exit { ‘TestResult == Pass’ { Eval ‘HVRes=1’; } } // exit if Pass 300mhz
Exec FunctionalTest { 
  PinsToTest ALL_PINS;
  IForce ‘-200mA’;
  ShortThreshHold ‘-200mV’;
  OpenThreshHold ‘-1.5V’;
  Category = 2;
}
Exit { ‘TestResult == Pass’ { Eval ‘HVRes=2’; } } // exit if Pass 200mhz
Exec FunctionalTest { 
  PinsToTest ALL_PINS;
  IForce ‘-200mA’;
  ShortThreshHold ‘-200mV’;
  OpenThreshHold ‘-1.5V’;
  Category = 3;
}
Exit { ‘TestResult == Pass’ { Eval ‘HVRes=1’; } } // exit if Pass 100mhz
  ‘TestResult == Fail’; // exit port 2
}

Test LV_FUNC {
Exec FunctionalTest { } 
Exit { 
  ‘TestResult == Pass’; // exit port 1, if true 
  ‘TestResult == Fail’; // exit port 2, if true 
}

Test STRESS {
Exec FunctionalTest { } 
Exit { 
  ‘TestResult == Pass’; // exit port 1, if true 
  ‘TestResult == Fail’; // exit port 2, if true 
}

Bin SWITCH {
  BinType Status;
  Exit { 
    ‘HVRes==1’; // exit port 1
    ‘HVRes==2’; // exit port 2
  }
'HVRes==3';   // exit port 3
}
}
