

# Extending STIL 1450 Standard for Test Program Flow

David Dowding<sup>1</sup>, Ernie Wahl<sup>2</sup>, Don Organ<sup>3</sup>

<sup>1</sup> Agilent Technologies, Inc. Loveland, CO. USA [dave\\_dowding@agilent.com](mailto:dave_dowding@agilent.com)

<sup>2</sup> Agere Systems Allentown, PA. USA [ejwahl@agere.com](mailto:ejwahl@agere.com)

<sup>3</sup> Inovys Corporation Pleasanton, CA. USA [don.organ@inovys.com](mailto:don.organ@inovys.com)

IEEE 1450 Web Site: <http://grouper.ieee.org/groups/1450/>

P1450.4 Working Group Web Site: <http://65.119.15.228/stil.4/>

## Abstract

*The IEEE P1450.4 Test Program Flow Extension to the STIL Standard provides a test program flow description that will allow full program generation by Design-to-Test automation tools, the portability and interchange of test programs between different ATE platforms, and enable greater reuse of program components. With the recent adoption of IEEE 1450 STIL[1] and its extensions, the test program flow extension will provide the semiconductor industry the means to greater automation and decrease the Time-to-Market aspects of new product introductions.*

## Introduction

There is little, if anything, in the design, test, and manufacture of semiconductors that has not been quantified by an economic impact of some measure. The overall expense in time and resources to develop, deploy, and maintain test programs has been found to be a large part of the cost to manufacture semiconductors. Design automation of test program generation has improved the economics of the IC design tasks. Great strides have been made toward automation of test pattern generation for both functional and structural tests. Automating test program generation has been addressed within larger test development organizations, however, this capability has been perceived as a proprietary advantage, and not made available industry wide. Providing a standard test program generation tool set would only serve a part of the need since each ATE platform

would require its own generator, parser and interpreter. The best initial step for aiding in the automating of test program generation is to provide a standard description of test program flow. This is the purpose for the P1450.4 STIL standard extension.

Part of the difficulty of developing a test program flow standard is that test programs have been in the industry a very long time. Each semiconductor manufacturer has developed standard test program practices and methodologies to manage ever increasing device complexities, and to serve their specific manufacturing flow requirements. Traditionally, ATE manufacturers have developed their test program compilers and environments. They have also provided methods to enable customer specific practices, improve program load and test time optimizations, as well as provide methods to manage new tester resources and innovations. Some of these ATE and manufactures' methods and practices have migrated throughout the industry and become widely adopted. The P1450.4 development effort has become a much broader specification in the attempt to be inclusive of most all of these methods and practices.

It is the charter of the P1450.4 Test Flow STIL Extension to support existing and new methodologies, standard practices and emerging standards. This specification is intended to provide a uniform test program flow description that can be layered within the 1450 STIL Standard. It should provide an interchange mechanism and a means for further automation between design tools and ATE platforms.

## Test Program Flow Extension Features

The P1450.4 extension work has wide scope. The following descriptions cover many of the features and capabilities of support. This is not an exhaustive list, but is intended to give insight into the methodologies, practices and data that are currently planned in this extension.

1. Provide complete alignment with the STIL 1450.0 and its extensions:

The same order and structures of the 1450-1999 STIL standard have been maintained. This enables the same parsing and processing methodology for STIL to be extended for test program flow data[2]. With the P1450.4 proposal, new program flow keywords and data constructs are being added, but the processing of current STIL will not require behavioral changes.

Just as with the base standard the top level P1450.4 extension constructs are blocked at the top level of STIL. As in STIL domain named blocks can be referenced. Unnamed blocks are considered as “default” for their type and the data contained within them is used whenever a construct of its type is in scope. This is identical to STIL data resolution behavior for named and unnamed blocks[3]. For example an unnamed SignalGroups block has global scope and named SignalGroups block is in scope while its referencing construct is in scope.

Consideration is being given to incorporated the IEEE 1450.2-2001 DC Level constructs seamlessly into the appropriate flow constructs[4]. Additionally, the P1450.4 extension plans where ever possible to use and build upon the P1450.1 and P1450.6 (EDA and Core Test Language) constructs.

2. Provide program sequence organization and control:

Today’s test programs are often not just straight line flows of tests. Some test programs have hundreds of tests suites. One of

the goals of this extension is to provide a manageable method to develop and maintain test programs with large numbers of test suites. Hierarchical grouping tests into “subflows” sequences and layering these groups within other “subflows” is part of the approach being defined. This hierarchical approach will provide support for libraries of reusable tests and test sequences. This extension is building constructs that allow this hierarchy. A given point in the flow description can represent a single TestMethod call or a complete subflow of several TestMethod calls with flow branching on test results and binning resolution. The highest view of the test program flow can be represented by a single “FlowNode” or test that can be “pushed into” to reveal lower levels of flow sequences that will ultimately return a “Pass” or “Fail” with “hard” and “soft” bin assignments.

The graphical representation of the flow data will be left to the implementation of tools operating on this data. The data itself will contain the structure and reference mechanism that will permit an test engineer familiar with the extension specification an intuitive conceptual view of the program flow.

3. Promote test program component reuse:  
As mentioned above in feature 1, test program flow blocks can be named. These test blocks can be defined such that they are complete test entities and can be referenced by their domain name. Once this object is placed into a given flow, it only need have its connectivity within the flow established (i.e. where it is in the sequence and where program flow control will go following the test.)  
Even ATE interrupt driven and real-time test system actions can be defined and named for reuse in various test programs.
4. Support test and test sequence modularity and hierarchical program organization

In addition to a test suite being named and then referenced multiple times, sequences of tests can be encapsulated into a “sub-flow” block and named. Libraries of these test sequences be used to contain the standard test methodology for device types or families of devices. These can then be used to build a test program flow. The sub-flow blocks can be called multiple times in a test program and reused across as many test programs.

The extension will support multiple test programs being fully specified in a given STIL file suite. Each program will be contained within its own TestProgram block. Each of these will have their own test interrupt and system actions (i.e. OnStart, OnReset, etc.). This would permit description for a single System-on-Chip (SoC) device separate test program descriptions for Wafer-Sort, QA/PackageTest, etc. within a single STIL file. Modules within these can be encapsulations for a complete sequence of tests, with only Signal resolution differentiation for test hardware and device packaging.

#### 5. Multi-Site control mechanisms:

Multi-Site methodology provides for better tester utilization, and providing for improved management of test costs. The impact on test program flow can in large part be dealt with in the tester operating system with little effect on test program impact. However, tester resources allocation and hardware sequencing dependencies can impose the need for multiple test sites generally running concurrently to have to switch to a site-serial execution. Further, the test flow sequence for failing sites may require special flow considerations. These “exceptions” must be anticipated in the test program flow. The program flow must provide tester directives for appropriate flow operation with regard to exception conditions. This extension has identified many of these flow situations and is providing mech-

anisms to specify these alternate flow directives. This is a complex domain of operation, and the extension may be limited in covering the basic situations, and leave others to the system control implementation of given tester platforms.

#### 6. Support general binning strategies:

Most ATE have the concept of “hardware” bins and “software” bins. The hardware bins relate to prober/handler bins or categorizing of the tested devices. There are typically a finite number of hardware bins available on ancillary test equipment. There may be several software bins associated with a hardware bin. The software bins are typically used for greater test result specification.

For example, a specific class of test may have several tests within the test program to test all pins and all aspects of the data sheet parameters for that test. One or more classes of test may use the same hardware bin, but the various tests may each have its own software bin to help isolate and record device test results.

The mapping of tests to their software and hardware bins is referred to as a “bin map”. This extension will define the bin mechanisms and provide constructs to support the use of these mechanisms. Further, the use of modular tests and sequences will allow reuse with proper bin mapping as a result.

#### 7. Alignment with test sequencing from EDA tools such as found in the P1450.6 Core Test Language:

The P1450.6 CTL structures use the P1450.1 Environment block to locate their CTL base structures[5]. This along with P1450.6 domain referencing mechanisms, a given “test methodology” can be encapsulated for each CTL supported core. Each core may have multiple methodologies, such as production test, diagnostics, etc. The test program combines and orders the various cores’ tests and test methodologies

with the test sequence and dependency information from the SoC designer.

The P1450.4 flow constructs can be used to align and sequence core tests within the overall test program. The core providers and the SoC device designers will use this extension to direct the testing sequences and provide for greater test intent content information to the test development group. It is intended that this extension, where ever possible, will achieve alignment with other STIL extensions to promote use and potential automations.

8. Provide for ATE system interrupts and assertions:

Asynchronous and real-time ATE system interrupts and assertions may vary from one tester platform to another. Specific device setups and interactions may need to be addressed with differing sequences of tester resource application. In the past, some of these special test actions have been placed in the test program and others placed outside the program, but associated with it. This extension provides a set of general interrupt and assertion type tasks, such as OnLoad, OnStart, OnReset, etc. These can be global tasks on a type and/or there can be named tasks associated with a type. The test program can declare named special task suites or use global user descriptions. In addition, the user may define User Defined tasks and associate them with test platform assertions. This approach provides the test program flow with complete description of all activities and sequences within and outside the traditional test program flow that are required for tester control.

9. Provide for a structured interface for communications with TestMethods:

This extension to STIL is focused on program flow and does not define the actual test suite or "TestMethod". To provide for required coordination between the test program flow and the activation, test result

assessment and response by the flow, an interface between the flow and test will be described. This interface description will provide a clear and sufficient data communication protocol for passing in parameters to the TestMethod from the flow and then receive results back from the test to the flow for correct follow-on flow response. A simple view of this interaction is analogous to a software function call that passes in parameters and a resultant code is returned to indicate the success or failure of the function. In addition to a Pass or a Fail result, some TestMethods may pass back to the test program flow a test result such a a measured value or an array of results. The interface to and from the TestMethods are accompanied in this extension with post-test actions that can be based on the test result(s) and an "arbiter" that evaluates the appropriate actions, such as binning assignments and device settings to conclude the test, or prepare the device for the next test suite in the flow, and concluding with the appropriate branching flow from this test suite.

10.Support multiple test behaviors in a single program or multiple test programs in a single STIL test suite:

As mentioned above, this extension provides mechanisms and constructs that have a modular nature. Referencing specific "named" constructs, can resolve device and tester resources and connectivity to provide tests that will run on a variety of ancillary test equipment, and provide test suites that are reusable and configured to the device or hardware settings.

For example, WaferSort and PackageTest suites may vary on the signals available at the boundary of the device. Some tests may be run in both cases with resolution of boundary signals being the only issue. In other cases, the test suite behavior will vary with the test condition type. Herein the extension provides the test developer flexibility to deal with differing test behavior

within the TestMethod or within the test program flow constructs. This provides for the flexibility to have multiple test programs in as many STIL file suites, or all test program behaviors in a single test program in a single STIL file suite, or some variation in between.

Other features not described above include but are not limited to the use of standard test system setup variables defined outside the STIL equation/spec set (such as program load determinations and flags). Run-time variables for system and program control. Multi-Site variable sets to support run-time variant behaviors. There is also specification to describe “in-line” test suites that are specific to a test flow and are not named or reusable. These and other capabilities are being developed in this extension to provide greatest test program flow and control description.

### **Working Group Collaborations**

The working group has been, and will continue to use all paths to collaborate with the other test flow standardization activities. Currently the working group is collaborating with the Open Test Architecture Consortium referred to as the “STC” to converge their Open Test Program Language (OTPL) and the 1450.4 extension specification to define a single industry specification for test program flows.

As mentioned earlier, this extension will leverage the description and named data variables for setup and run-time test status and results established in the “SEMATECH Testing-Equipment Specific Equipment Model (TSEM)”, chapter 8. These variables and test results report parameters are derived from an operational model for testing equipment as viewed by a factory automation controller.

Additionally, the working group is evaluating existing ATE “STIL-like” flow methodology

and construct syntax, as provided by those ATE platforms. These provided syntax sets have been used as a starting point for the P1450.4 flow specification syntax. It is the plan of these ATE platform providers to adapt their system’s flow syntax to that of the P1450.4 syntax as it is adopted. There are other ATE providers that are engaged in the working group efforts and will then utilize the standard as their test program flow language in future releases of their tester’s software.

### **Anticipated Adoption**

In addition to the ATE adoption plans mentioned in the previous section, there are semiconductor manufacturers with in-house tools and commercial EDA Automatic Test Program Generators (ATPG) that can be modified and extended to produce full test programs using this specification’s test program flow syntax.

For test development teams that do not rely on ATPG generated test programs, they will be able to share and maintain libraries of ATE control routines, TestMethods, BinMaps and hierarchical test sequences and flows. With these, the test development effort will be dramatically improved.

IP Core test methodologies will expand their offerings from patterns and timing to include levels and complete test sequences. Designers and EDA integration tools will enable and generate the desired test flow directives to automate the Design-to-Test tools to generate complete test programs.

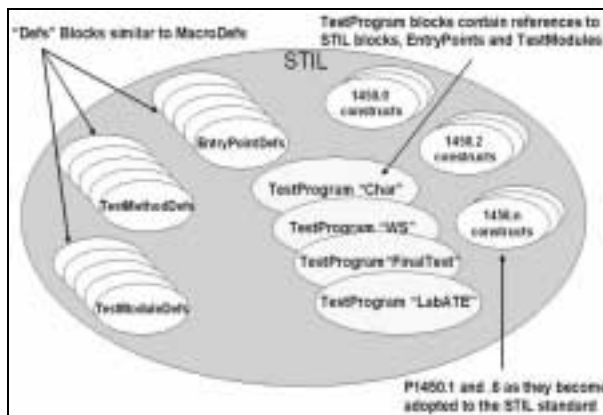
Adoption of this extension will provide significant portability benefits to test organizations that support test facilities with multiple tester types. Test development in general will be improved as ATE vendors provide fully STIL and STIL extension compliant systems.

## Test Program Flow Conceptual Model

Use case descriptions and conceptual models have been used in the development of this extension. The conceptual models provide a way to communicate the essence of the extension constructs and their interactions in the test flow. The following is a top-down view of portions of the model with some descriptions and definitions.

Figure 1 shows a graphical representation of the TestProgram blocks within a STIL file. TestModule/FlowModule definitions, and the tester system entry point definitions (interrupts and assertions routines) blocks are contained in a MacroDefs-like containers. These are at the top level of the STIL file and are outside the one or more TestProgram blocks.

FIGURE 1. Flow Constructs within a STIL File



These named TestProgram blocks contain the complete test program declarations and references to define the test program flow, sub-flows, tester interrupt/assertion routines, binning solutions and test references and definitions describing a complete test. Test flow sequence is defined by the interconnection of the hierarchical flow and test objects.

The following is a partial list of the construct blocks that compose the flow components within the conceptual model and are indicative of keywords of this extension:

## TestProgram Block:

The top level test program construct, of which there can be one or more within a STIL file. One may be global (unnamed). There may be one or more named TestProgram blocks in a STIL file.

## Flow Block (or TestFlow):

This is the top level program flow construct. There can be one unnamed Flow block. There can be one or more named Flow blocks. This block contains definition of flow and bin entities that make up a given test program flow.

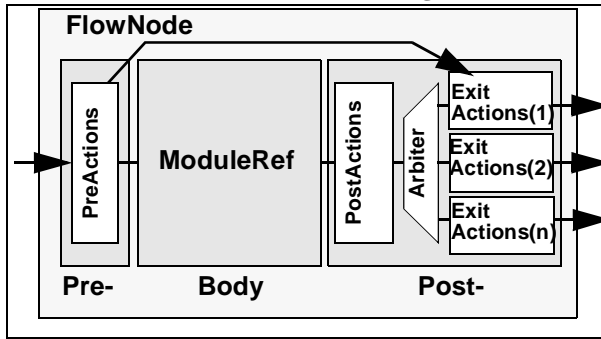
## EntryPoint:

This is a reference to a special program level task activated by the tester (tester operating system, system interrupts, etc.) This entry point references a TestModule. There is a general set of EntryPoint entities defined by this extension (i.e. OnStart, OnReset, etc.). These can be named and one instance of each can be unnamed and treated as global to any Flow block that does not declare a named one of each type. When a flow is active and a tester event requires an EntryPoint, the associated EntryPoint TestModule/FlowModule is run.

## FlowNode:

A node in the program flow that contains a ModuleRef or "Body" that references a TestModule or FlowModule. (See Figure 2.) This node has PreActions that define the entry point into the node and may contain actions, declarations such as Spec/Category selection, etc. The Post section contains PostActions, Arbitrator, and ExitActions. The ExitActions give directives as to the follow-on flow path taken out of the FlowNode. Absence of actions in the Pre or Post sections may cause default actions to be used

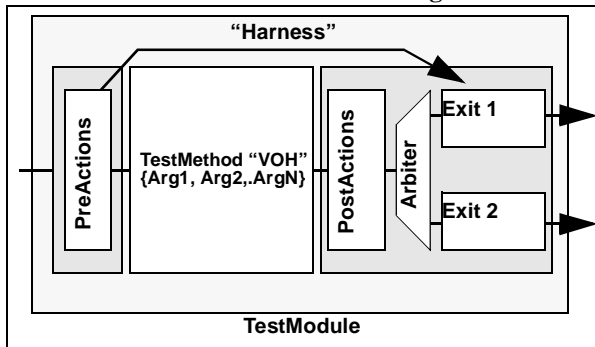
**FIGURE 2. FlowNode Block Diagram**



**TestModule:**

The module referenced in a FlowNode ModuleRef. This module is defined in the ModuleDef block and has a domain name. The instantiation of the TestMethod (i.e. VOH) can be “in-line” or “defined-before-use”. “Define-before-use” is the mechanism by which two or more FlowNodes can refer to the same “Test Object” (i.e. Module). Test Module instantiation involves placing the instantiated TestMethod inside the “harness” (the harness is the grey portion of the box labeled TestModule.) (See Figure 3.)

**FIGURE 3. TestModule Block Diagram**



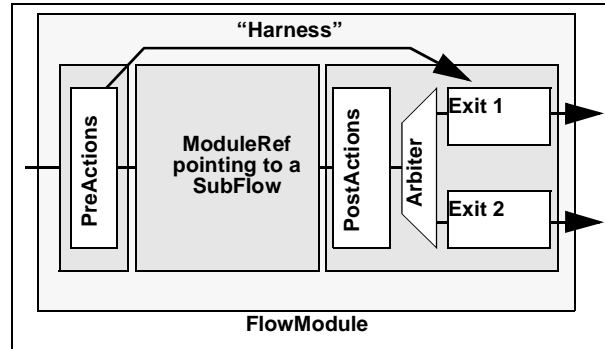
Test Module Example:

TestMethod “VOH” { Arg1, Arg2,...ArgN }  
 is the type definition, and  
 Test VOH { Arg1, Arg2,...ArgN }  
 is the instantiation of the VOH TestMethod  
 in the test module.

**FlowModule:**

The FlowModule is similar to the TestModule but differs in that the FlowNode ModuleRef references a “subflow” that is predefined in the TestModuleDefs block. (See Figure 4.)

**FIGURE 4. FlowModule Block Diagram**



**BinNode and BinMap:**

The BinNode has two natures: terminal and flow-through. The terminal BinNode type is a “terminal or end point to flow execution. The flow-through type provides the capability to flow to a into and out of a BinNode to accomplish the activities described in the BinActions and then to proceed to a following FlowNode.

**TaskNode and DecisionNode:**

These are non-test type nodes used for non-test activities such as flow decision content.

**Overview of the Proposed Syntax**

At the time of this writing, the syntax for the P1450.4 extension’s constructs and data are still being defined from the collaboration with the STC and ATE syntax proposals. This effort is scheduled to be concluded and resulting in the Draft proposal document prior the presentation of this paper. Syntax and examples will be provided at that time.

## **A Brief History of P1450.4 Development**

The P1450.4 PAR effort began in late 1998. The P1450.2 DC Levels, P1450.4 Test Flow and P1450.5 Test Methods working groups were essentially the made up of the same participants.

This extension has taken longer than the original PAR approved time frame due to three factors.

1. Work between the three sister extensions serialized their work due to available members and dependencies on the other extensions' constructs. In March 1999 it was decided to suspend work on the P1450.4 to complete the P1450.2 DC Levels definition.
2. Industry economic conditions restricted travel and work force resizing reduced the ability to meet and the candidate members for the working groups.
3. In March, 2002 the P1450.4 resumed its activities. Significant test program flow technology and EDA/DFT mechanisms had changed since suspending the extension. The working group leveraged some work from the initial working group, but a majority of the conceptual model an syntax had to be reworked.

The working group is now leveraging from the P1450.6 Core Test Language extension and from other industry standards, such as the SEMATECH Testing-Equipment Specific Equipment Model (TSEM) for production test variables[6]. Additionally, we are endeavoring to deal with a variety of current test techniques, such as Multi-Site test flows and concurrent or parallel testing techniques.

## **Limitations**

At the time of this writing, limitations of this extension are still being identified and

resolved. This list of issues and resolutions is comprised in a document the working group maintains during its meetings and communications.

## **Conclusion**

The inclusion of P1450.4 Test Program Flow Extension to the STIL Standard will provide for a complete digital test program specification. This extension to the STIL standard will provide the semiconductor industry with a standard interchange vehicle for digital test programs and flows. EDA test generation tools and ATE tester operating systems will have a standard center point on which to converge their outflow and input.

This convergence will provide for greater Time-to-Market achievements due to automation of the test generation process. It will be a large factor in reducing test costs by removing the need for future test program conversions between different ATE platforms from different vendors. Test engineering resources can focus more of their precious tester time on device testing and manufacturing issues and less on porting and maintaining test program code.

The adoption of this extension will, of course take time, but with each step of adoption the semiconductor manufacturers and EDA and ATE vendors will achieve greater levels of test automation resulting in earlier test development and better tester utilization.

## **Acknowledgements**

The following individuals are active members of the P1450.4 working group:

Dave Dowding (Agilent), Daniel Fan (Credence), Yuhai Ma (Advantest), Tom Micek

(Motorola), Jim Mosley (Credence), Eric Nguyen (Advantest), Jim O'Reilly (Agilent), Don Organ (Inovys), Jose Santiago (Philips), Douglas Sprague (IBM), Tony Taylor and Ernest Wahl (Agere).

- [5] Tony Taylor, "Standard Test Interface Language (STIL), Extending the Standard", Proceeding of the International Test Conference, 1998
- [6] SEMATECH Testing-Equipment Specific Equipment Model (TSEM), Technology Transfer # 96063139A-STD, July 31,1996

## References

- [1] IEEE Computer Society, IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data Language Manual IEEE Std. 1450.1999. IEEE New York, September 1999.
- [2] Gregory A. Maston, "Structuring STIL for Incremental Test Development", Proceedings of the International Test Conference, 1997.
- [3] Tony Taylor, Gregory A. Maston, "Standard Test Interface Language (STIL) - A New Language for Pattern and Waveforms", Proceeding of the International Test Conference, 1996.
- [4] IEEE Computer Society, IEEE Standard for Extensions to Standard Test Interface Language (STIL) (IEEE Std. 1450.1999) for DC Level Specification. IEEE New York, September 1999.