Open Architecture Software for OPENSTAR™
Test Platform

Yuhai Ma
Advantest America, Inc.
3201 Scott Boulevard
Santa Clara, CA 95054

Abstract

A new concept of Open Architecture Automated Test Equipment (ATE) is being discussed hotly in the semiconductor industry in terms of new hardware and software architectures. This paper presents Open Architecture Software for OPENSTAR™ — the Open Semiconductor Test Architecture that is being defined by the Semiconductor Test Consortium. The software architecture and related components of the software are discussed in the paper.

1. Introduction

Today, the ATE industry is facing a technological revolution, in which a new Open Architecture ATE is emerging, which is competing with and will eventually replace traditional ATE. This new concept of Open Architecture ATE is becoming a very hot topic that is being discussed in the industry [1][2]. In order to define the new open architecture ATE to solve the challenges of cost-effectively testing complex semiconductor devices including System-on-Chips (SoCs), System-in-Packages (SiPs) and other complicated devices, the Semiconductor Test Consortium (STC), Inc., an industry-wide initiative, is formed among ATE users, ATE vendors, and module vendors. The STC is now defining the specification of the Open Semiconductor Test Architecture — OPENSTAR™ and has released the first draft of the specification. This paper presents the OPENSTAR™ system architecture, discusses its software architecture and components, and describes the test programming and development environment as well as application programming interfaces.

2. OPENSTAR™ System Architecture Overview

The OPENSTAR™ test platform provides reconfigurability and scalability to be capable of coping with various test requirements and preventing tester obsolescence [3]. The OPENSTAR™ system architecture provides support for multiple hardware implementations and can be conceptually envisioned as a distributed system as illustrated in Figure 1 [4]. Each test site is envisioned as dedicated to testing a single device under test (DUT), and functions through a configurable collection of test modules. Each test module is an entity that performs a particular tester function. For example, a test module could be a device power supply, a pin card, an analog card, etc. This modular approach provides a high degree of flexibility and configurability. For example, a collection of sixteen 64-pin modules could be
configured into eight test sites to serve as separate, independent units to test eight separate 128 pin-count DUTs, or into two test sites for two 512 pin-count DUTs, or one site for a 1024 pin-count DUT.

In each configuration, the test site is under the control of a single Site Controller (SITEC). Each type of test module supports a particular, standard interface that enables the Site Controller to communicate with it. This standardization of the communications interface, as well as inter-module communications and connectivity to chassis allows for a high degree of plug-and-play between conforming modules from different vendors. Each Site Controller could be deployed on its own dedicated CPU, or as a separate process sharing the same CPU with the System Controller (SYSC) and/or other Site Controllers. The communication between a Site Controller and a module-set could be provided by a variety of connectivity enabling hardware, referred to as the Module Connection Enabler, as long as it serves as a high speed bus for fast data transfer (for loading pattern data, gathering response data, and providing control, etc.).

Figure 1. OPENSTAR™ System Architecture
This architecture allows for unhindered scaling up as per-site test complexity increases, or the number of independent test sites increases. The central System Controller, with limited responsibilities for test station functionality, is not as taxed as in a system where the central System Controller has responsibilities for managing all test site functions. With most of the test station functionality being relegated to the Site Controllers — thus allowing independent test site operation — the System Controller serves as the overall system manager. This coordinates the Site Controller activities, manages system-level parallel test strategies, and provides for handler/probe controls as well as system-level datalogging and error handling support.

3. OPENSTAR™ Software Architecture

The OPENSTAR™ Software [4] project is aimed at delivering a modular ATE solution that scales well and is extremely flexible. The OPENSTAR™ Tester Operating System (TOS) software is a distributed system that has components deployed across the System Controller and the Site Controllers. The TOS has two principal operating modes; online and offline. The former implies actual tester hardware, while the latter provides system hardware emulation. Figure 2 shows a high-level representation of the system.

The System Controller software is the primary point of interaction for a test engineer in a verification and/or debug environment. It provides the gateway to the Site Controllers, and synchronization of the Site Controllers in a multi-site/DUT environment. User applications and tools — graphical user interface (GUI)-based or otherwise — run on the System Controller. The System Controller could also act as the repository for all test plan related information, including compiled test plans, compiled patterns, test conditions or parameters files, etc.

The Communications Library provides the mechanism to communicate with the Site Controllers in a manner that is transparent to user applications and test programs. The Standard Interfaces provide open interfaces to the OPENSTAR™ framework objects that execute on the System Controller. Tools/Applications allow interactive and batch control of the test and tester objects, through GUIs or other means. The applications may make use of the scripting interfaces as well.

The OPENSTAR™ Software supports tester configurations with Site Controllers being responsible for running one or more DUTs, i.e., test sites. Hence, in a system comprising multiple Site Controllers, each Site Controller controls testing of one or more DUTs, executing the test engineer’s test programs. For each site it controls, it provides high-level synchronization of the test modules corresponding to the DUT. It is important to note here that one should not equate a test site with a Site Controller. A single Site Controller can control one or more test sites, and purely from a software architecture point of view, there is no restriction on the number of sites controlled by a single Site Controller (apart from usual computer system resource considerations).

Module Commands Implementation classes are provided by the module hardware vendors, offering interfaces for test plans to access hardware modules, and interact with common tester hardware components as well as with other test-related objects. The Backplane Communications Library provides the interface intended for standard communications across the software backplane, thereby providing the functions necessary to communicate with the modules connected to that site.

The Modules provide hardware components to support device testing, such as digital tester channels, device power supplies, or parametric measurement units. Module software controls a particular instrument hardware module.

The user-level software resides and runs on the System Controller. User management and security issues are treated with the exact same facilities as are provided by the computer operating system, which considerably simplifies these
tasks. The System Controller also handles remote display, and, together with the Site Controllers, provides support for offline services, with tester emulation to support such tasks.

**Figure 2. OPENSTAR™ Software Architecture**

### 4. Test Programming and Development Environment

#### 4.1 Test Programming Environment

The principal component of the OPENSTAR™ test programming environment is the test plan. A test plan is a test program written by the test engineer. The test plan uses test classes that realize the separation of test data and code for particular types of tests, which provides good reusability of code.

A test plan may be written directly as a C++ test program, or described in a set of test plan description files. These description files use the OPENSTAR™ Test Programming Language (OTPL) [5], and are processed by the OTPL Compiler to produce C++ code. The generated C++ code can then be compiled into the executable test program. The data required for populating a test class instance, such as levels, timings, etc., are specified by the user in the test plan description files.
The OTPL defines the syntax and semantics of files that provide the input for a test program. One of the objectives to be met in the design of this language is modularity. A language supports modular development if it permits users to write individual components dealing with different aspects of the test, and then permits these components to be mixed and matched in various ways to yield a complete test program. To this end, the OTPL allows for the information needed for a test program to be assembled together from several files.

4.2 Bridging Design to Test

For a very long time, our industry has wanted to knock down the wall between design and test. EDA (Electronic Design Automation) vendors and ATE vendors have been struggling to deliver benefits to the product development cycle, but in the meantime, EDA and ATE have contributed to building a higher wall between design and test.

[2] pointed out that, for the future, a further growth is foreseen in the importance of test-related software systems. Both the complexity of the product designs and the complexity of the process technologies drive the need for even tighter integration between Design For Test (DFT), Design For Manufacturing (DFM), production test control/management, and diagnostics/failure analysis. ATE has to evolve towards more flexible, open hardware and software architectures that allow for rapid re-configuration with a variety instruments that are able to collect a wider range of manufacturing data.

[1] stated that the only practical solution is an open architecture allowing any designer of a specialized component to design the necessary instrumentation with its use enabled by the infrastructure of Open Architecture ATE. Both the ATE architecture and the EDA systems must enable test strategy partitioning between BIST, other DFT approaches and traditional ATE. The closed architectures and one dimensional EDA tools that dominate the landscape today will not be viable tomorrow. The Test technology vendors should work towards a unified and open infrastructure to design and develop tools for the test tool market [6].

The OPENSTAR™ software is intended as open architecture software that allows users or software vendors to take advantage of what is provided and extend it for their own needs. By developing against OPENSTAR™ Application Programming Interfaces (APIs), which will be described in the following section, the OPENSTAR™ software architecture allows the users or vendors to develop additional software components for the system, at the level of test classes, applications and tools. This provides both EDA systems and ATE platforms with a practical solution to establish an effective communication link between each other, and share a common ground by adoption of industry standards, such as IEEE STIL [7], Core Test Language (CTL) [8][9], and IEEE P1500, Standard for Embedded Core Test (SECT) [10][11], etc.

5. Application Programming Interfaces

5.1 User API

The user API provides interfaces and procedures used to create additional software-only capabilities for the OPENSTAR™ software framework [12]. Here is a brief description of these user APIs.

Core API
The core API provides the capabilities to extend the OPENSTAR™ test language in two ways; 1) the high-level (OTPL) language may be extended through the use of pre-header files in the OTPL language; and 2) the C++ constructs that are used to create the Test Plan may be extended through direct use of C++.

**Datalog API**

The datalog capabilities provided by the OPENSTAR™ system allow for data collection and tracking of test status while executing tests and test plans. The OPENSTAR™ core functionality provides an implementation of a datalog capability. This may be replaced with an implementation, for example, STDF, provided by the user or other software developers. The datalog framework contains a Datalog Manager to handle the various datalog objects in the system.

**Simulation Framework DUT Modeling API**

In order to run the software in an offline test mode (in the absence of tester hardware), it is necessary to have a simulation model of the DUT. The user may create a DUT model in C++, or as a Verilog model running on a Unix server. A configuration file is used to integrate the various simulation pieces into the OPENSTAR™ offline simulation framework. The loadboard is simulated as well, and a configuration exists for this component, providing connections between the device and the tester simulation.

**Tools API**

The Tools API is the set of interfaces against which tools and applications are written. This includes GUI applications as well as scripting tools. The Tools API is designed as a set of proxies on the SYSC that reflect behaviors on the SITEC. The application/tool developer writes against the proxy implementations, the information is transferred via a communications layer to the SITEC and the specified action is carried out. Any required data are transferred bidirectionally between the SYSC and SITEC via the communications layer. The purpose of the Tools API is to provide a conceptually simple, extensible set of interfaces that developers can use for the implementation of tools and scripting applications. The Tools API is applicable to both user (software-only) and vendor level development. The interactions with the methods of the API are the same in both cases.

5.2 **Vendor API**

The vendor API provides interfaces and procedures used to create software modules necessary for the support of new hardware modules [13]. When new hardware is to be created for an OPENSTAR™-based system, there is a requirement for software modules to support the hardware. These include modules for calibration and diagnostics, and simulation of the new hardware. The following is a brief description of these vendor APIs to be used for vendor software module development in support of hardware.

**Standard Interfaces for Module Control**

The OPENSTAR™ standard interfaces comprise the basic set of functionalities exposed by the OPENSTAR™ system to the developer. This portion of the interface provides the basic hooks for common test-related items such as tester pins, device power supplies, etc. Developers of new hardware need a mechanism for communication with both their actual hardware and the offline emulation. These connections must integrate with the OPENSTAR™ framework so that these pieces of the system may be called during normal operation of the software.
Backplane API

Developers of software modules that will be used to support new hardware will make use of the capabilities of the system backplane. This includes the functions for developing against the backplane as well as configuration and diagnostics. The interfaces for the backplane are substantially similar in the offline and online environments. This results in only small modifications to code when running in these two different modes.

Digital Module HLC API

The low-level support for control of Digital Modules (DMs) is mediated through the High Level Command (HLC) API. These are the interfaces used by the OPENSTAR™ implementations of such functions as TesterPin in order to access the underlying data.

Device Power Supply HLC API

The low-level support for control of Device Power Supplies (DPSs) is mediated through the High Level Command API. These are the interfaces used by the OPENSTAR™ implementations of such functions as PowerSupply in order to access the underlying data.

Diagnostics/Calibration/Debug Development API

Diagnostics and calibration is a critical part of a semiconductor test system. For any new hardware, means for calibrating the hardware as well as diagnosing problems must be provided. The structure for the integration of the calibration and diagnostics routines into the OPENSTAR™ framework specifies the interface methods that the developer must implement in order to have the routines properly executed.

Pattern Compile Manager/Loader Development API

New Digital Module hardware requires a mechanism for compiling and loading pattern information. This is handled by a custom pattern compiler developed by the vendor, and managed by the OPENSTAR™ Object File Manager. A minimal set of interfaces must be implemented in order for the Pattern Compiler to be properly invoked from the OPENSTAR™ system.

Simulation Framework Module Modeling API

Module vendors need to provide a software simulation of their module for use in offline simulation/emulation mode. The simulation framework provides the approaches for the module developer to link their module emulation into the OPENSTAR™ offline system. The API provides interfaces for event access, processing and handling. The module emulation developer will provide an implementation for these functions.

6. Conclusions

This paper presents a new Open Architecture ATE platform — OPENSTAR™, and the STC is defining the specifications for its hardware and software architectures. In the paper, we have described the OPENSTAR™ overall system architecture, presented Open Architecture Software for the OPENSTAR™ test platform, discussed its test program-
ming and development environment as well as application programming interfaces. The OPENSTAR™ architecture allows ATE developers and hardware and software module developers to concentrate on developing new test solutions and innovating breakthrough technologies for our industry. OPENSTAR™ provides a practical solution for the semiconductor industry to timely offer diversities of required testing functions, prolong the life of ATE, and eventually solve the challenges of cost-effectively testing complex semiconductor devices including SoCs, SiPs and other complicated devices.

References


