

# A Roadmap for Boundary-Scan Test Reuse

Gene Wedge  
Tom Conner  
VICTORY Software Development  
Teradyne, Inc.

International Test Conference  
October 1996

## Abstract

This paper proposes a Layered Model for boundary-scan testing to help identify opportunities for standardization. Serial Vector Format [1] and an accompanying Application Programming Interface, developed for the Application Specific Electronic Module (ASEM) project sponsored by ARPA, are presented as examples of a novel approach to the representation of standards for the ATE industry.

## Introduction

Successful implementation of boundary-scan testing requires cooperation among users, equipment manufacturers, and tool vendors. The IEEE 1149 family of standards is laying the foundation for that cooperation [2, 3, 4]. Before any of these standards were developed, boundary scan was practiced by a few vertically integrated companies in separate, proprietary implementations.

The adoption of IEEE Std 1149.1-1990 and IEEE Std 1149.1b-1994 for Boundary Scan Description Language (BSDL) set the stage for ATE companies to cooperate with CAD companies and chip foundries in providing circuit assembly manufacturers with the infrastructure for implementing boundary-scan tests and diagnostics. These standards allow the separate pieces of the infrastructure to be developed by separate companies, each applying its expertise to those parts of the problem with which it is most skilled.

To facilitate the continuation of standards development, this paper presents a model for the process of generating boundary-scan tests and diagnostics. The model is composed of four layers of abstraction, with the structure of the assembly-under-test at the top and the actual patterns applied to test it at the bottom. The model enables us to identify points in the process where

standardization might be beneficial and to focus efforts on implementing new test and diagnostic techniques. This model is described in detail in the next section.

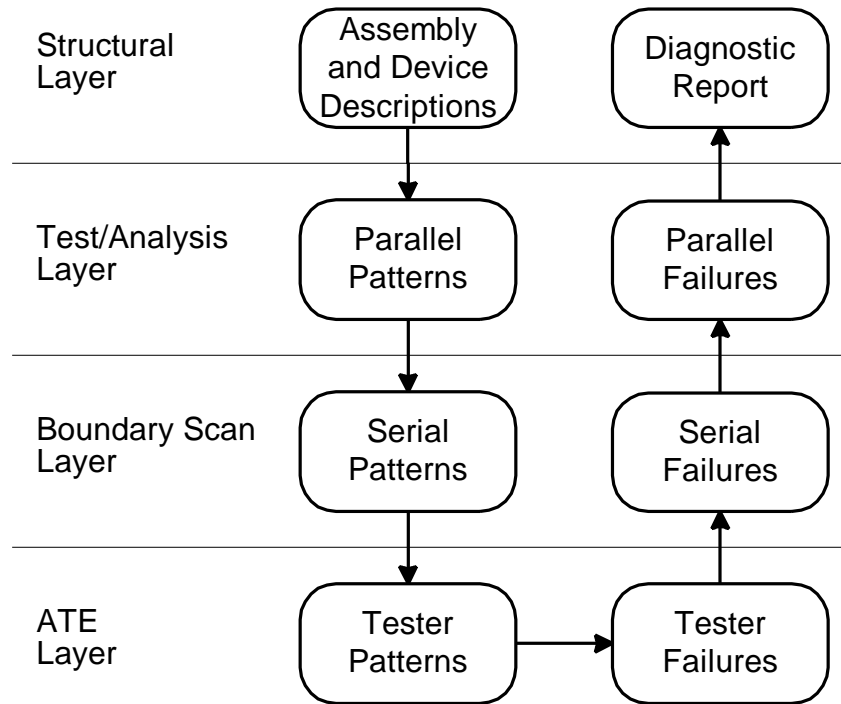
Serial Vector Format (SVF) is an example of an information-exchange medium that is benefiting from the standardization process. SVF has been used for several years by a number of companies to express serialized boundary-scan test patterns for a wide variety of ATE architectures. As part of the ARPA-sponsored ASEM project, we have developed an Application Programming Interface (API) that implements the data model represented by SVF, encapsulating the specialized knowledge of the format specifics. This enables a developer with no detailed knowledge of the SVF format to quickly implement a translator to convert SVF into a tester-specific format, with a very low risk of error in interpreting the SVF format. We present this approach, the sharing of APIs, as model for the development of future information-exchange standards.

## The Layered Model

For purposes of conceptualization, implementation, and debugging, it is useful to divide up the boundary-scan test and diagnostic problem into separate levels of abstraction (see Figure 1 for a data view of the model).

The layers descend from a human-oriented view of the assembly to be tested down to the actual patterns that are applied by an ATE system. The testing problem begins with a description of the assembly-under-test, such as a netlist and various descriptions of the components of the assembly, including BSDL models for the boundary-scan components. This is shown in the Structural Layer of the model. The data at this level is in terms of the assembly and its components, without reference to testing.

**Figure 1. Data view of layers.**



The test engineer divides up the circuit into testable units, depending on the test access (including “virtual access” provided by boundary-scan cells) and the test techniques and tools that are available. Then the engineer uses some test-generation process to produce parallel test patterns for each portion of the circuit. Patterns may come from a test library for components of the circuit, from simulation of portions of the circuit, or from ATPG techniques.

So far, none of the work necessarily involves boundary scan, except to keep in mind those points in the circuit where boundary scan provides test access into the circuit. The test engineer is now working in the Test/Analysis Layer.

For boundary-scan testing, these parallel patterns are then serialized. The bits from the parallel patterns are mapped into the appropriate scan cells and the shifting of the scan cell data is coordinated with the application of any parallel stimulus or response through physical tester channels. At this Boundary Scan Layer, the test is represented as serial patterns.

These serial patterns are then translated to be run on a particular tester, which brings us down to the ATE Layer. This translation step is separated from the serialization

step so that the serialization software can concentrate on the boundary-scan aspects of the problem without knowledge of the tester where the patterns will ultimately be run.

When failures are detected, they are typically reported in some form native to the particular tester. If we convert these tester failures to some serial failures format, then we can use generic diagnostic software, which again is independent of the detailed nature of the tester. In the Boundary Scan Layer, the serial failures are deserialized, or mapped back to the device leads represented by the failing scan cells, to produce parallel failures, which are located in the Test/Analysis Layer.

Once we are back up in this domain of parallel patterns and failures, we can perform diagnosis, again without reference to boundary scan or to the target tester. The diagnostic process analyzes the parallel failures with reference to the structure of the circuit to produce statements about faults in the assembly, in terms of its components.

This entire model can also be viewed in terms of the processes that take place at each layer (see Figure 2). In this view, we also include data that are used to communicate from the test generation side of the drawing

to the diagnosis side. At each level, data produced by the test generation process are needed by the corresponding diagnostic process.

These data are isolated from the other layers, describing only those aspects of the entire process that are relevant to their particular layer. The scan vector data describe the structure of the scan vectors for the purpose of deserialization, without regard to the tester in the layer below or to the particulars of the parallel patterns in the layer above. The nature of the diagnostic data depends on the type of test—interconnect test, component test, or cluster test—again, without any reference to the boundary-scan aspects of the test. The topology data describe the original assembly and its parts in terms that will be used to finally express the faults diagnosed. All these pieces of data may be taken together as a diagnostic database for a given test of an assembly on a given tester.

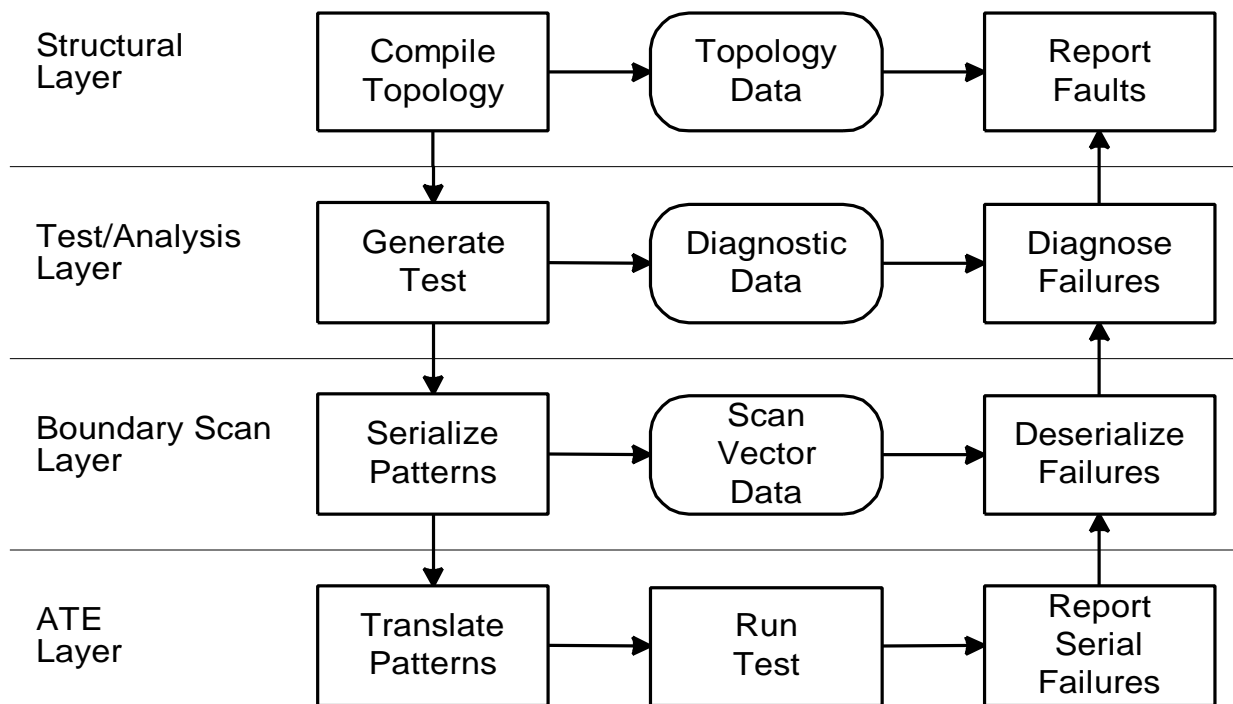
Portions of the model may be grouped together in a single program. Deserialization, diagnosis and reporting, for example, are implemented in a single diagnostic engine in the work we performed under contract to ARPA. Even so, the internal architecture of this software reflects the distinction between the layers of the model. This has allowed us to plug in new diagnostic algorithms at the

Test/Analysis Layer, without disrupting the surrounding software.

Specifically, we implemented diagnostics for testing clusters of non-scan parts surrounded by boundary-scan components by integrating the LASAR fault-dictionary algorithm into the diagnostic engine [5, 6]. Users can now fault simulate non-scan clusters without reference to boundary scan and feed the resulting patterns and fault-dictionary data to boundary-scan test and diagnostic software. Furthermore, the resulting tests can be run and diagnosed on a variety of ATE platforms, given the portability afforded by separation of the ATE Layer. This ARPA work has been commercialized and is now available with the current release of Teradyne's VICTORY software.

The entire Layered Model may be implemented by a single ATE vendor, or separate companies can use their particular areas of expertise to implement individual layers. In order for such cooperation to be successful, even within a single company, there must be an agreed-upon format for the data to be shared among the separate processes. If a data format at a given layer is to be widely shared, then at least a *de facto* standard is necessary.

**Figure 2. Process view of layers.**



This approach offers several advantages to end users. Component software and hardware can be purchased from different vendors. Or a tailored package can be purchased from a third party who pulls together components from various implementers. Most basically, a fully deployed set of standards based on a test development model makes it possible to accomplish tasks larger than any one organization is willing to undertake.

### Opportunities for Standardization

Figure 3 shows all the potentially relevant categories of data in the Layered Model whose representation might be standardized. In our development of a portable boundary-scan test generation and diagnostic infrastructure, we have developed or used existing formats for all these categories, some of which we see as ripe for standardization. These are shown in bold.

In the Structural Layer, there are many widely used representations for assembly and device descriptions. Among the relevant standards that exist at this level are EDIF and BSDL. At the bottom level, each particular ATE system will have its own representations, so standardization is not envisioned there.

Portability to other vendors' testers, as targeted by our contract with ARPA, drove us to focus on the serial pattern and failure data at the Boundary-Scan Layer. We

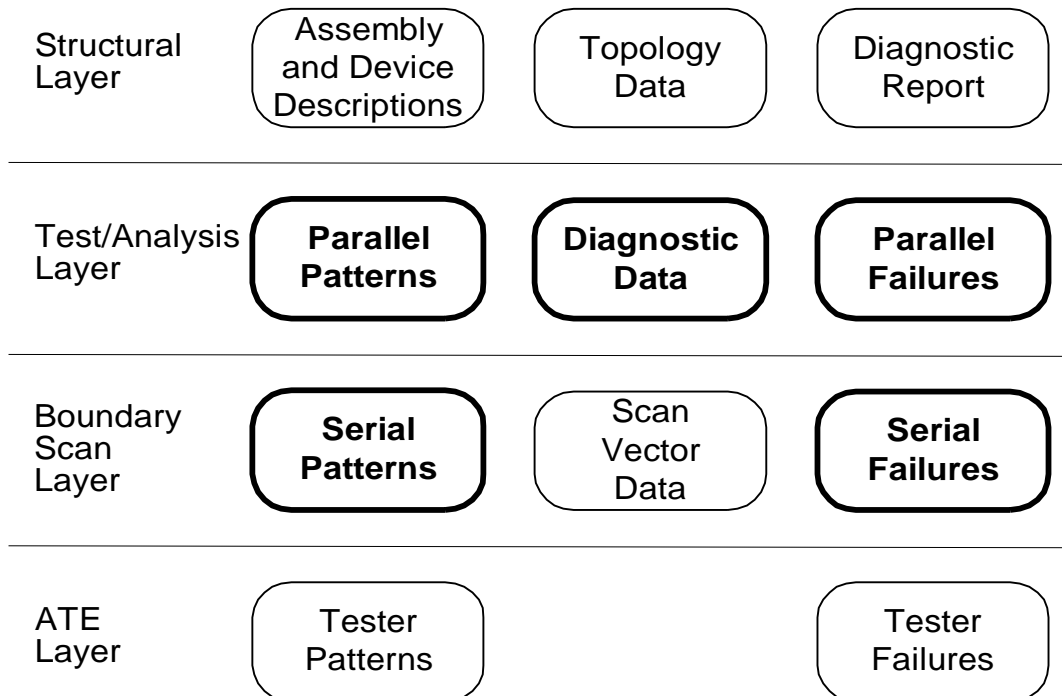
provided interfaces for both sets of data between the Boundary-Scan Layer and the ATE Layer.

For representation of serial patterns we use SVF, which is being considered by members of the IEEE 1149.1 working group as a candidate for standardization. In the next section, we discuss our work in interfacing between SVF serial patterns at the Boundary-Scan Layer and tester patterns at the ATE Layer.

On the other side of the model, for enumerating serial failures, we have published the Digital Test Failure Data (DTFD) format [7]. As with SVF, we provide an API to enable rapid porting of the DTFD writer to a variety of ATE systems. While the scan vector data also lie in this layer, this is not as important to standardize, since the serialization and deserialization schemes are relatively tightly coupled and, therefore, would probably be performed by software of common origin.

We see the Boundary-Scan Layer as the prime target for standardization efforts in the immediate future. With SVF and DTFD and their associated APIs, we have been able to implement a boundary-scan test generation and diagnostic scheme that is easily portable to virtually any digital test system. It would be of great benefit to the test community to formalize some form of standard at this level.

Figure 3. Opportunities for Standardization.



We have also exchanged data formats with third-party developers at the Test/Analysis Layer. In particular, there is work under way to generate test vectors for non-scan clusters using surrounding boundary-scan access points. The parallel patterns can be serialized independently, and the diagnostic data are expressed in the LSRTAP fault-dictionary format, which directly feeds the diagnostic engine produced by our current work under contract to ARPA. There is an effort underway with IEEE Std 1029 to standardize the diagnostic data format for this type of test as well as a parallel failures format.

### ARPA ASEM Effort

A major portion of our effort in the ARPA-sponsored ASEM project focused on the transitions between the Boundary-Scan Layer and the ATE Layer (see Figure 3). The problem was taking serialized test vectors to a specific ATE and returning failure data from that ATE for diagnosis. This allows an ATE user to combine test generation and diagnostic software from one vendor with an ATE system from a different vendor. To state the problem more concretely, our task was to provide a mechanism to permit serial vectors expressed in SVF, the *de facto* standard interchange format for serial vectors, to be run on any tester, and another mechanism to permit the tester's failure data to be retrieved for diagnosis. Here we focus on the test vector side of the ASEM project, which was referred to as Portable Serialization.

SVF describes 1149.1 Test Access Port (TAP) operations, such as the shift of an entire scan vector. For example, the following SVF statement specifies that the TAP controllers are to be sequenced into the Shift-DR state, applying 19 bits of data at TDI and expecting 19 bits of data at TDO. The drive and detect data are expressed in hexadecimal in the `tdi` and `tdo` parameters. The `mask` and `smask` parameters can be used to mark some `tdi`

and `tdo` bits as don't-care; in this example, all bits are marked as relevant.

```
sdr 19 tdi(3e241) smask(7ffff)
      tdo(7f31a) mask(7ffff);
```

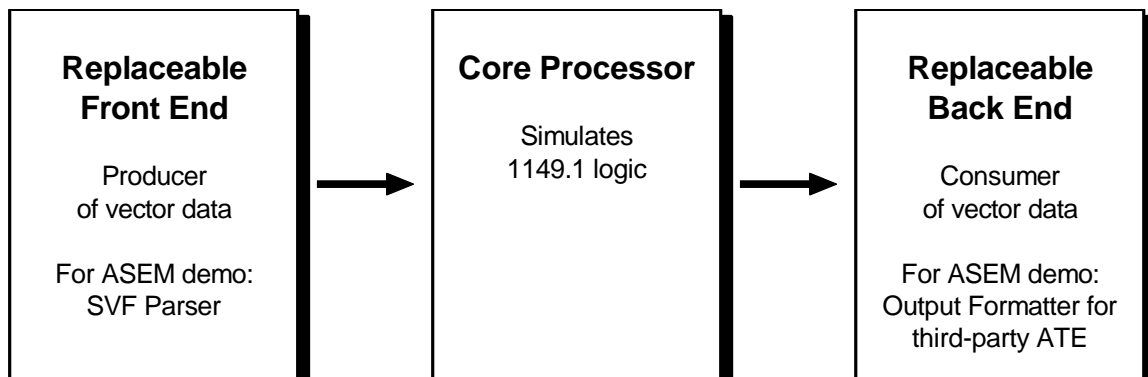
SVF also provides a statement to specify drive and detect states on parallel tester channels. SVF does not specify details of the TMS states needed to traverse the TAP controllers or of parallel pin timing; these details are left to the implementation. Our work yielded an SVF reader that served out vector data through an API. However, the SVF format itself is not central to our solution. SVF can evolve or be replaced by another format without obsoleting support for third-party ATE. Figure 4 shows how we partitioned the problem.

The Front End is the producer of vector data. The Front End implemented for the ASEM demo was a parser for SVF Revision C.

The Core Processor accepts vector data from the Front End. It simulates the operation of TAP controllers, generating proper TMS drive states and appropriately coordinating assertion and testing of parallel tester channels. Thus, the Core reduces the boundary-scan vectors to a simple stream of raw test patterns.

The Back End is the consumer of this raw pattern data. A major thrust of our ASEM work was to present a well-defined API to make the creation of Back Ends as simple as possible. For the ASEM demo, an independent consultant familiar with the target third-party ATE implemented a Back End. This consultant, who had previously completed a similar converter project without the benefit of our API, estimated a 60% saving in implementation time using the API. A Teradyne engineer used the API to integrate SVF into a new tester product in less than 3 days, using about 100 lines of code.

Figure 4. Serial Vector Processor Architecture.



Both the Front End and the Back End are easily replaceable. In the future, we could implement a new Front End to parse a future revision of SVF. Alternatively, the Front End need not be an SVF reader at all. We could attach the serializer program that currently writes the SVF file as a Front End, thus avoiding the intermediate SVF file altogether.

Because the Core has packaged the scan data as raw patterns, the Back End author may ignore boundary-scan details and concentrate on the details of the target ATE. The Back End need not be a simple writer of an ATE patterns file. For example, we have a Back End that directly calls the runtime API of a tester, bypassing the ASCII pattern file and applying the vectors immediately. Yet another Back End generates logic simulator files to be used as a test bench for design verification.

The products of our ARPA ASEM work were a Front-End SVF reader and a Core Processor with API support for Back Ends. To facilitate the implementation of Back Ends, we documented the interface between the Core and the Back End and provided sample Back Ends for different types of vector data consumers.

We implemented these components as dynamic or shared libraries on PC and UNIX platforms. The choice to use shared libraries allows any of the three parts to be replaced independently. This permits easy updating of the Core in the field without relinking Back Ends. Plus, it allows different Front Ends to be used interchangeably for different projects, so that new projects can use new formats without disturbing old projects.

A full discussion of the API between the Core and Back End is beyond the scope of this paper. What follows is a basic overview of the central functions that a Back End would typically create and use.

For Back Ends that will generate patterns for ATE without built-in 1149.1 support, the Core Processor serves out patterns on a clock-by-clock basis. In this case, the Back End author implements a function called TckFalling. The Core calls the Back End's TckFalling function each time the Core simulates a falling edge of TCK. The TckFalling routine then calls a routine in the Core Processor to determine drive states of TMS and TDI and expect data for TDO, as well as parallel tester channel drive/detect states. The Back-End code can then format that data in the language of the target ATE.

Back Ends targeted at 1149.1-aware ATE generally need the drive/detect data for each scan but not the TMS signals for TAP controller state sequencing. The Core calls such a Back End only once per scan operation. Authors for these Back Ends implement functions called ScanData and ScanInstruction. The Core calls ScanData

for each data scan or SDR command in the SVF file. ScanInstruction is called for each instruction scan. The implementation of ScanData or ScanInstruction calls another Core routine to retrieve two arrays of 1s, 0s, or Xs—one array for TDI and one for TDO.

Clearly, this discussion is too brief to give a good sense of using our API to create a Back End. However, full details are available by downloading the VICTORY Vector Converter SDK (Software Development Kit), the product of our ARPA ASEM Portable Serialization work, from Teradyne's World Wide Web site:

<http://www.teradyne.com/cbt/prodctr/pvict.html>

### **A New Way of Sharing Data Models**

Our work brings APIs to the test industry as a new way of understanding, specifying, and standardizing a data model. Choosing to use an API as the vehicle by which a data model is documented, shared, and standardized has benefits for the framers of the standard as well as for the eventual user. The standards committee can ignore syntactic details of ASCII languages and leverage an already specified programming language whose behavior is already well understood. While implementing to an API standard, the user avoids all concern over syntactic details and gets straight to the task at hand, that of processing the data itself.

While our experience has shown that sharing APIs through dynamic libraries is an excellent new way to standardize on data models, this mechanism is not without problems. Portability among computing platforms is important; however, there are a lot of non-portable approaches that can inadvertently be taken with dynamic libraries on any one platform. Furthermore, interfaces must be exported via language-specific header files, making it awkward at best—and impossible at times—for code written in a different language to call the API.

The software industry has benefited from the sharing of APIs and is moving toward distributed object technologies that provide the same benefits, yet overcome the problems described above. Methodologies such as CORBA and OLE provide much more crisp and portable definitions for interfaces. These new technologies enable components to work together easily even if implemented in a mixture of different languages and operating in a heterogeneous computer environment.

### **History of SVF**

There are lessons to be learned from the experience of SVF. In the fall of 1991, SVF was created by Teradyne, Texas Instruments, and Tektronix to facilitate the exchange of serial vector information. Since then, there

have been two minor revisions to the format, neither of which represented any radical change or major addition of information.

More recently, there has been a more formal effort towards standardization of SVF. An unofficial study group drawn largely from the IEEE 1149.1 working group has been working on a specification for an enhanced SVF since mid-1994. This effort has not yielded any results to date. There are two difficulties with this new SVF effort. First, there is a strong temptation to add information into the new format from the Test/Analysis Layer and the Structural Layer, thus complicating the data model and making it more difficult to represent. Diagnostic data, device instruction data, and chain topology data are all being considered for addition. The second difficulty is common to any effort to standardize an ASCII format: simply that it takes a lot of hard, detailed work to yield a solid specification for a language that is concise and easily parsed.

## Conclusion

The standards process is a long and difficult road. The IEEE Std 1149.1-1990 took 5 years to reach standardization. The BSDL standard in IEEE Std 1149.1b-1994 took 4 years, and the IEEE Std 1149.5-1995 MTM-Bus (Module Test and Maintenance Bus) standard took about 5 years. However, standardization is well worth the effort because it allows each company to specialize in its own area of expertise and to cooperate with the efforts of other companies.

The progress of recent boundary-scan standardization efforts illustrates that one must start with a clearly chosen data model that focuses on a small, well-bounded portion of the overall boundary-scan testing problem. Meanwhile, we must maintain a clear vision of the overall problem so that the portion we choose yields—over the long term—the most opportunity for cooperation. We have proposed a high-level model that can provide the clarity to define appropriately scaled portions of the standardization problem.

Boundary-scan testing has long held out the promise for testing hierarchical assemblies from ICs to MCMs to boards to backplanes to systems. And boundary-scan techniques promise to play an increasing role throughout the product life cycle, from prototype development through manufacturing to depot test [8]. As the design and ATE communities work to fulfill these promises, the need for cooperation through component hardware and software can only become more pressing.

Our ARPA ASEM work has demonstrated the benefits of sharing the serial vector data model via an API and

shared libraries. Having worked through the issues surrounding this mechanism, it is clear to us that such distributed object technologies as OLE and CORBA promise new ways to do an even better job of sharing standards, allowing more concentration on the data model and more flexibility in implementation.

## References

- [1] Texas Instruments, Inc. and Teradyne, Inc., "Serial Vector Format Specification, Revision C," Texas Instruments, Inc., September 1994
- [2] "IEEE Standard Test Access Port and Boundary-Scan Architecture," *IEEE Std 1149.1-1990 (Includes IEEE Std 1149.1a-1993)*, IEEE Standards Board, 345 East 47<sup>th</sup> Street, New York, NY, October 1993.
- [3] "Supplement to IEEE Std 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture," *IEEE Std 1149.1b-1994 (Supplement to IEEE Std 1149.1-1990 and IEEE Std 1149.1a-1993)*, IEEE Standards Board, 345 East 47<sup>th</sup> Street, New York, NY, March 1995.
- [4] "IEEE Standard Module Test and Maintenance (MTM) Bus Protocol," *IEEE Std 1149.5*, IEEE Standards Board, 345 East 47<sup>th</sup> Street, New York, NY, July 1995.
- [5] D.E. Wedge, P. Hansen, "Test and Diagnosis Environment," *Proceedings ARPA Electronic Packaging and Interconnect Conference*, 1995.
- [6] D.E. Wedge, "Simulation, Fault Dictionaries, and Boundary Scan: Testing and Diagnosing Logic Clusters," *Proceedings of IEEE Autotestcon*, 1996.
- [7] Teradyne, Inc., "Digital Test Failure Data Language," VICTORY 2.30 Reference Manual, 1996.
- [8] P. Hansen, "Hierarchical Design: Taking Boundary Scan to the Next Level," *Proceedings of IEEE Autotestcon*, 1996.