Considerations in the Design of a Boundary Scan Runtime Library

Terry Borroz
System Test Group
Teradyne, Inc.
North Reading, MA
terry.borroz@teradyne.com

***Abstract*—IEEE 1149.1 boundary scan has become a widely used test technique since its introduction in the 1990s. Such tests are typically developed using software and hardware provided by companies that specialize in boundary scan testing. To help integrate this type of testing into large-scale military test systems, the idea of a boundary scan runtime library was introduced in 2011 and is now beginning to be adopted. Such a runtime library allows each boundary scan supplier's runtime software to apply previously-developed tests using the general purpose digital hardware in a large-scale military test system, even though those tests were initially developed using different hardware.**

Keywords-Boundary scan; JTAG; IEEE 1149.1; Deep Serial Memory

I. INTRODUCTION AND BACKGROUND

Boundary scan (often called "JTAG") is a technique in which special standardized circuitry is included in an IC to facilitate testing and data transfer. It was standardized in the 1990s as IEEE standard 1149.1 [1]. To meet the standard, an IC must provide a "test access port" for data communication (the "TAP") and registers that allow driving and capturing digital signals at the pins (the "boundary") of the IC. The TAP has only 4 (optionally 5) wires, transmits data serially, and is designed so that this serial data transmission can be daisy chained through all the boundary scan ICs on a board. Thus all of these ICs can be accessed from a single board-level TAP using only 4 or 5 wires.

Since its introduction, boundary scan has become widely used for detecting and diagnosing basic device pin faults, shorts, and opens in digital circuitry during manufacturing test. Electronic assemblies are now routinely designed to support boundary scan testing. At least a half dozen companies provide boundary scan test generation and runtime software tools, generally accompanied with proprietary hardware to deliver the tests. One of these vendors is typically selected during the design or prototyping phase of product development. Later, manufacturing test engineers usually attempt to reuse this design effort, which entails using the software and hardware supplied by the tool vendor. The hardware typically consists of a small boundary scan controller managed by a PC. This hardware controller usually delivers the tests through an interface pod that needs to be close to the TAP port of the unit under test.

This development oriented test environment can cause problems on large-scale systems used for Defense and Aerospace test because the various units under test (UUTs) come from multiple design groups, each of which may have selected a different boundary scan tool supplier. To support this variety of UUTs, the path of least resistance would be to require the production test system to include test hardware from each of the boundary scan vendors associated with the target UUT designs, as shown in figure 1.

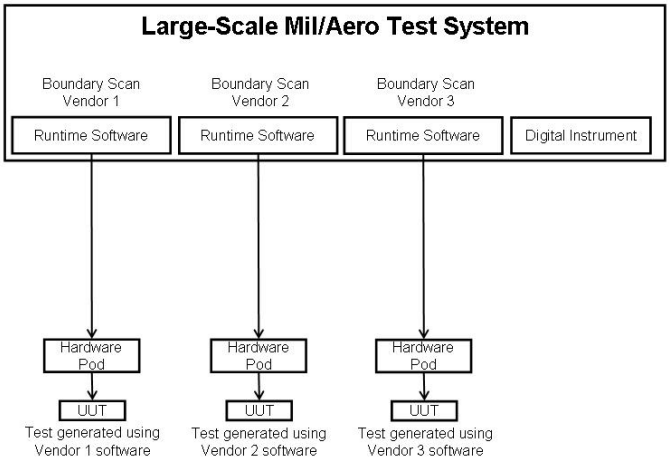

Figure 1. Fixturing with separate hardware from each vendor

This is straightforward, but it is complex and redundant. It includes many pieces of hardware that do the same thing. The pods must all be near the UUT, thus competing for the same space in a fixturing environment that is often already very tight. This situation becomes even worse when considering boundary scan tests that also connect to parallel I/O signals, since now the independent number of connections to be fixtured is potentially much larger. The result is that designing each boundary scan vendor's hardware into the fixture produces an unacceptably expensive system integration and logistics support situation. Figuring out a way to use a single set of boundary scan hardware would help with this issue.

Two years ago a runtime library was proposed to address these issues [2]. A runtime library addresses this problem by providing a unified way to apply boundary scan tests developed by different vendors, as shown in figure 2.

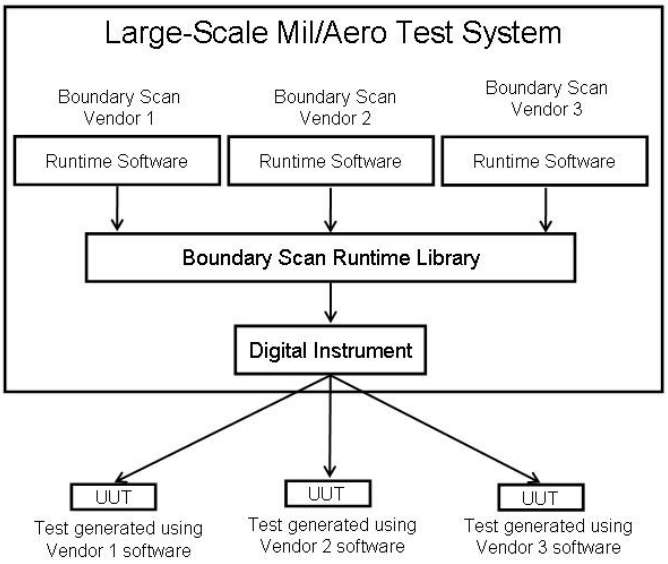

Figure 2. Fixturing with a boundary scan runtime library

Such a system would execute boundary scan tests using the general purpose digital instrumentation commonly found on large scale test systems. These digital instruments are well suited to boundary scan testing. Some provide features such as deep serial memory that optimize such operations [3]. They can also easily handle massive amounts of parallel I/O. A boundary scan runtime library for such a system would provide a standardized API for these capabilities. Boundary scan vendors could then add a relatively small amount of code to their existing software suites that would execute their tests using this runtime library as an alternative to their proprietary hardware. Since proposing such a system in 2011, we have worked toward implementing these ideas. The rest of this paper discusses some of the considerations related to such an effort.

II. API Issues

The overall organization of a system using a boundary scan runtime library would be as shown in figure 3.

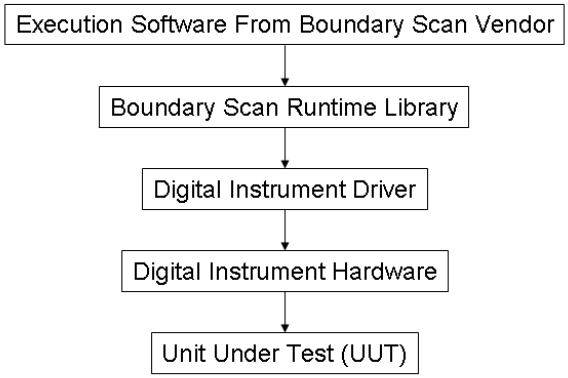

Figure 3. Overall organization with boundary scan runtime library

One of the most important issues in the design of any runtime library is the Application Programming Interface (API) that it exposes. This is critical because it defines how the runtime library appears to its clients. In this case, the clients are developers who write code to execute boundary scan tests by calling the boundary scan runtime library.

Ideally, the API would primarily speak in terms of common boundary scan activities at the UUT, such as shifting and state navigation. Calls from the client to perform these activities should require little knowledge (preferably none) of the details of the underlying execution hardware. The API should support a full range of boundary scan activities, including support for multiple TAPs and for specification of related activity on parallel pins. There should be a way for parallel pin activity to be coordinated with boundary scan output changes as a TAP port navigates through the update state. If possible, it would be good to support arbitrarily many TAP ports and parallel pins (but ultimately limited by available tester pins, of course).

A boundary scan runtime library client would begin by defining connections to the UUT (TAP ports plus parallel pins). He would then navigate the TAP ports, perform shifts and/or parallel pin operations, and request results as desired. The boundary scan runtime library would ideally keep track of the state of each TAP controller in the UUT, so that clients who wished to could simply request that a TAP be navigated to a specific state, letting the runtime library take care of the details. In many cases, a client might also want some control over test parameters such as TCK frequency and voltage levels. (But note that the capabilities to modify these parameters could be greatly simplified relative to the fine grained control that is often provided by an underlying general purpose digital test instrument.)

Note that this conceptual view aligns pretty well with the existing SVF language. SVF is an interchange format, not an API, but the level of abstraction is very similar. This is fortunate, because most boundary scan vendors and users are already familiar with SVF.

Also note that specifying entire shifts with a single call is more efficient than alternatives that would operate at a more granular level. It is efficient because it minimizes the total number of calls. It also allows potential optimizations for speed over an entire shift that wouldn't be possible otherwise. (This is an example of the well known "chatty vs. chunky" tradeoff, which is widely discussed in software literature [4].)

III. Range of Test Types and Sizes

Boundary scan tests vary widely in size and complexity. A boundary scan runtime library should be able to support the entire range of possible test types. Figure 4 shows the range of sizes of some representative boundary scan execution sequences.

Test Type	Purpose	# of data shift operations	~ # of shift clocks (TCK, shift cycles)	Uses Parallel I/O
TAPIT (TAP Integrity Test)	Check 1149.1 path prior to using it for primary tests	1	1K – 10K	No
BICT – Boundary In-Circuit Test	Check for pin-levels faults, opens, shorts around 1149.1 devices surrounded by bed-of-nails	2	1K – 50K	Yes
VIT – Virtual Interconnect Test	Test for pin-level faults, opens, shorts in interconnects between 1149.1 device pins (optionally including edge I/O)	15 - 50	150K – 500K	Maybe
VCCT – Virtual Component & Cluster Test	Pin faults for conventional devices surrounded by 1149.1 devices in EXTEST	100 - 1000	500K - 5M	Maybe
	Program a conventional 256K X 8 bit FLASH memory surrounded by 1149.1 devices in EXTEST	1.25M (5 cycles per word)	2.5B	Usually No

Figure 4. Approximate sizes for various test sequences

In general, increased size results in increased time of execution. The execution times are often not significant for the smaller items in figure 4, but execution of the longest sequences on unoptimized hardware can take so long that execution speed overrides all other considerations.

The shorter items in figure 4 require detailed reporting of every failure so that faults on the UUT can be accurately diagnosed. The longest item can usually get by without this since a simple yes/no answer about success or failure is usually sufficient for flash programming. This mitigates the execution time issue somewhat, but not enough to make a big difference.

Using a system that includes a boundary scan runtime library can sometimes be helpful in situations where test execution is too slow. This is because the runtime library naturally divides the entire system into two independent pieces, which allows the implementer of the slower piece to improve his piece's performance without involving the developers of the other piece.

IV. Range of Execution Hardware

Thinking about the hardware that is available now (and extrapolating slightly into the future), one can classify digital test hardware into three broad levels of support for boundary scan execution. These are roughly in order of increasing efficiency of boundary scan testing:

1. A basic general purpose stored program tester. This can apply arbitrary sequences of drive and expect values to a collection of digital pins, but with no support to optimize for those pins that service boundary scan TAP ports.
2. Same as level 1, but with additional hardware to optimize interaction with TAP ports, such as deep serial memory to efficiently store long serial bit sequences.
3. Special purpose hardware specifically designed for boundary scan.

Simple testers (level 1 above) have been used for years, and level 2 testers are available and beginning to see more use. Level 1 is sufficient for small tests, but as tests get bigger and take longer to execute, the speed improvements that come with increasing complexity become more and more important. Level 3 testers are only speculative now, but they could be designed and might become available in the future.

A boundary scan runtime library could potentially support all of these kinds of underlying digital test hardware (and others). In view of this diversity, it makes sense to design the boundary scan runtime library with a modular architecture, so that plug in back ends to support new hardware types can be added as new hardware becomes available.

In addition to architecture, hardware can also vary in the boundary scan clock rates ("TCK rates") that it can support. Boundary scan is often thought of as something that's used at very low speeds. Basic boundary scan tests like interconnect tests are short and don't benefit much from high TCK rates, so they have traditionally been run at very low TCK rates, often at around 1 MHz. Also, especially in the early days, adding boundary scan at the board level was not a top priority for board designers, so they weren't careful about laying out the TAP signals on the board to support high clock rates.

Consequently, especially in some older designs, the design of the UUT will not support high TCK rates.

However, there are also some situations in which much higher TCK rates are possible and desirable. These typically occur in cases like flash programming where large data streams make throughput very important. It is not uncommon to encounter cases in which board designers have been careful and TCK rates in the tens of MHz can be achieved.

Digital test instruments have traditionally been dedicated custom designed hardware, but some reconfigurable test instruments are starting to become available [5]. Potentially, such instruments can be especially attractive for throughput critical boundary scan applications like flash programming. If such an instrument were accessible through a boundary scan runtime library, you could potentially configure the instrument for high throughput boundary scan, then do whatever flash programming was required, then reconfigure it to do the digital tests for which it was mainly acquired. If the reconfiguration is fast enough (say within a few seconds), then the reconfiguration time could easily be recouped by the improved performance during flash programming. A runtime library that supported a runtime defined instrument that could be reconfigured specifically for boundary scan would be an especially powerful combination.

REFERENCES

[1] IEEE Standard 1149.1-2001, “IEEE Standard Test Access Port and Boundary-Scan Architecture”.

[2] Borroz, T. "Using General Purpose Digital Hardware To Perform Boundary Scan Tests" IEEE AUTOTESTCON 2011.

[3] Marchetti, T. and Borroz, T. "Programming Flash Memory with Boundary Scan Using General Purpose Digital Instrumentation" IEEE AUTOTESTCON 2010.

[4] Voluminous web discussion can be found by searching for "chatty vs chunky", for example "Chatty vs. Chunky: A Developers' Guide", http://www.idevnews.com/stories/61, accessed July 2013.

[5] Teradyne, "High Speed Subsystem", http://www.teradynedefenseaerospace.com/products/high-speed-subsystem, accessed July 2013.