Using General Purpose Digital Hardware To Perform Boundary Scan Tests

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

Abstract—**Boundary scan tests are typically developed to make use of the proprietary hardware provided by one of the companies that specialize in boundary scan testing. This can lead to fixturing and logistical problems on large scale test systems used for Defense and Aerospace test, because testing many different assemblies whose tests rely on different boundary scan vendors can require the use of test hardware from each of those vendors. This paper presents a better approach, in which a common runtime software library can be used by each boundary scan supplier to apply his tests using the system's existing general purpose digital hardware.**

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

I. Introduction

IEEE 1149.1 Boundary Scan was introduced in the 1990s and has become an invaluable methodology 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 usually 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.

This development flow 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, test systems are required to include test hardware from each of the boundary scan vendors associated with the target UUT designs. This produces an unacceptably expensive system integration and logistics support situation.

This problem can be addressed by using a common runtime software library that can apply boundary scan tests using such a test system's existing general purpose digital hardware. Boundary scan suppliers can then update their runtime software to use the library as an alternative to proprietary hardware. This would apply the boundary scan tests using digital hardware that is already present in the test system, and could greatly simplify fixturing.

II. Boundary Scan Overview

Boundary scan is a technique in which special standardized circuitry is included in an IC to facilitate testing and data transfer. Boundary scan is often called "JTAG" because it grew out of an effort by a working group called the "Joint Test Action Group". For the most part, boundary scan was initially intended to facilitate testing for board-level production faults, and thus to provide an alternative to traditional bed-of-nails in-circuit testing.

This early JTAG work eventually led to the IEEE 1149.1 standard for the IC circuitry that enables this technique. The current version of this standard is IEEE Std 1149.1-2001 [1], and there are now several related standards [2].

The 1149.1 standard requires that a compliant IC 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 such 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.

The TAP signals consist of a clock (TCK), incoming and outgoing data lines (TDI and TDO, respectively), a mode signal to control the serial data transmission (TMS), and, optionally, a reset signal (TRST). In a typical configuration, TCK, TMS, and TRST are bussed together while TDI and TDO are chained from IC to IC. The TDI pin of the first device in the chain is the principal data input. The TDO of the first device is connected to the TDI of the second device. The TDO of the second device is connected to the TDI of the third device, and so on. The TDO of the last device in the chain is the principal data output. The slowest device in the chain limits the clock speed for all devices in the chain [3].

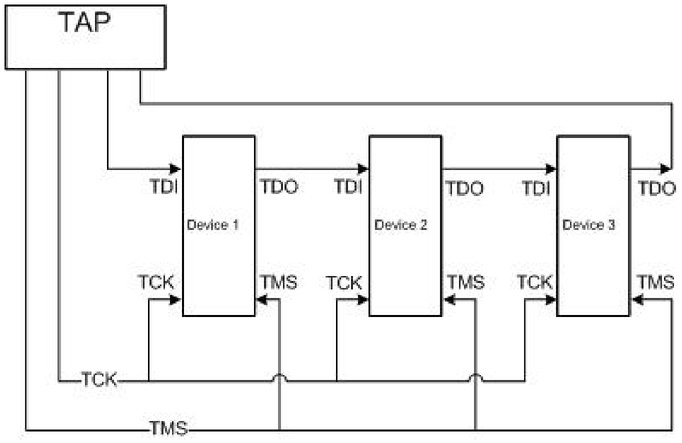

Figure 1. Boundary Scan Chain

Boundary scan operation is controlled by a state machine in each IC. This state machine is defined by the 1149.1 standard. State traversal depends on the TMS value seen by an IC at each rising edge of the clock. All boundary scan ICs in a chain are always in the same state during normal operation. Note that the same TMS signal is typically connected to all ICs in a chain.

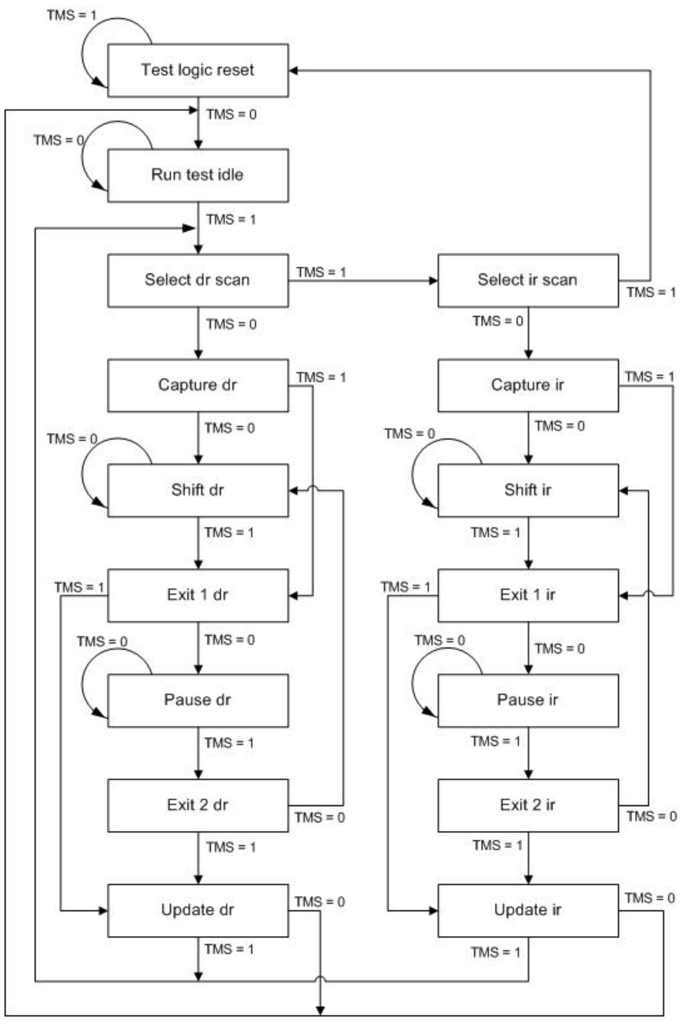

Figure 2. TAP State Machine

The initial reason for this TAP port infrastructure was to access a register connected to the IC's external pins for board test purposes, but there can be other registers as well. These are called "data registers". Each IC also has a single instruction register which controls a mux used to insert any one of the data registers between TDI and TDO.

The TAP state machine has states by which data can be scanned into the instruction register, or into the currently selected data register. There are mandatory instructions for obvious test applications. Each selects an appropriate data register for subsequent data shifting operations.

Besides these mandatory instructions, an IC designer can also define instructions that select other data registers, which can support whatever additional operations the designer wishes to implement. Over the years, this capability has been exploited to use 1149.1 data transmission in many new applications, such as programming FPGAs and supporting debugging in microprocessors and DSPs.

III. Development of Boundary Scan Tests

Almost all boundary scan tests are developed using tools provided by a handful of small vendors that specialize in boundary scan testing. These companies typically provide software for test generation, test debug, runtime execution, and runtime diagnostics. Each vendor also supplies low cost hardware for benchtop use exclusively with their own software. 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 UUT.

These systems are widely used for test program development and engineering prototype debug. Most large companies use software and hardware from multiple boundary scan vendors.

IV. Problems in the Production Test Environment

In the Defense and Aerospace industry, production testing is usually done on large scale test systems which test a large variety of UUTs. This usually means that each production test system must run boundary scan tests that were developed using the tools from many different boundary scan vendors. One way to run such tests would be to design hardware from each of these boundary scan vendors into the production test system, as shown in figure 3.

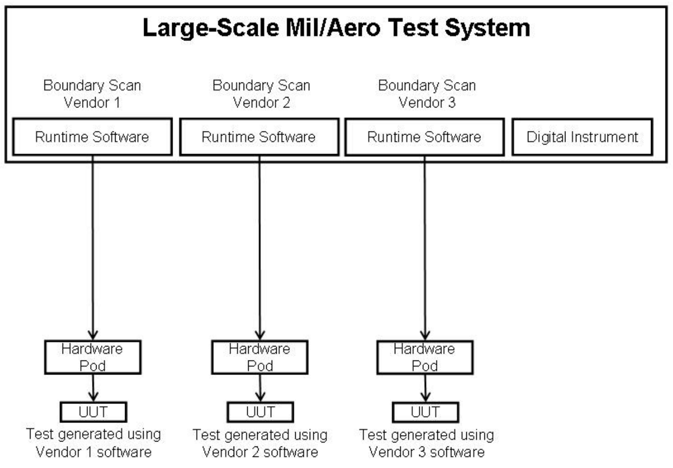

Figure 3. Fixturing with separate hardware from each vendor

This is undesirable for a number of reasons. First of all, it is complex and redundant (and potentially wasteful) to include many pieces of hardware that do the same thing. The pods are especially challenging in this respect because they must all be near the UUT, thus competing for the same space in a fixturing environment that may already be very tight. In fact, in many cases, the designers of the production test system would like to dedicate a specific location in their UUT fixturing for TAP signals. In this case, using multiple pods would require somehow multiplexing them to this single TAP connection. Figuring out a way to use a single set of boundary scan hardware would help with all of these issues.

Another issue is that benchtop testers provide poor support for the massive parallel I/O associated with functional test. Benchtop boundary scan systems typically provide some support for parallel I/O by using additional boundary scan devices to synthesize parallel signals that can then be connected to the UUT. But exporting this technique to the large scale test environment is even more onerous that supporting multiple pods. The production testers already have tester channels connected to the I/O pins of each UUT, but benchtop software can’t control them.

In addition, benchtop testers generally lack the robust pin electronics (in terms of programmability and protection) that have long been standard in the production test environment.

V. How These Problems Could Be Addressed

Many of the above problems could be improved if there were a way to unify the application of boundary scan tests developed by different vendors. In the past, there have been some attempts to provide capabilities similar to this. The Serial Vector Format (SVF) language [4] has long provided a way to describe the application of an entire boundary scan test. The Standard Test And Programming Language (STAPL) [5] later provided a similar capability aimed at using the boundary scan TAP port to program programmable devices.

However, these languages fall short of what is needed because they support batch-oriented techniques that don't allow for interaction between the software of the boundary scan vendor and the behavior of the UUT while the test is being run. Thus they are not suitable for the common situation in which the boundary scan software wants to examine results while applying a test in order to diagnose. Boundary scan vendors also provide interactive development and debug tools that require immediate control and result reporting. These tools also need a system that supports interactivity.

One way to address these problems would be to make a runtime library that executes boundary scan tests using the general purpose digital instrumentation commonly found on large scale test systems.

VI. Relevant Characteristics of Digital Instruments

These digital instruments are very generic. Within wide limits, they can synthesize arbitrary sequences of activity and test for arbitrary responses across a very large array of pins (potentially hundreds). They typically support widely programmable drive and expect voltages on all pins. They also provide very flexible timing, with programmable clock rates and sophisticated driver formatting and programmable edge positioning. Digital test instruments with this much flexibility can execute almost any type of testing, including boundary scan tests. It would be possible to run boundary scan tests on such a tester using its native API. However, there are several reasons why it makes sense to use a special runtime library optimized for boundary scan.

First of all, the generic nature and extreme flexibility of such hardware means that the native APIs of these test instruments are necessarily complex. This flexibility is useful in general but much of it isn't needed for boundary scan applications. Thus, for purposes of addressing boundary scan testing, it would be helpful to have a simplified API for boundary scan users that would hide the unused programmability in such a system.

In addition, note that the basic organization of such test instruments gives equal resources to all pins. This organization is not optimal for boundary scan. This has been noted in the past [6]. Some manufacturers have added features that address these issues, such as deep serial memory, which allows very deep memory to be dedicated for use by a handful of pins that require it [7]. This fits in very well with boundary scan, which requires very deep data streams on the TDI and TDO pins while the rest of the tester does very little [8]. A runtime library would be a perfect place to take advantage of these features.

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, as shown in figure 4.

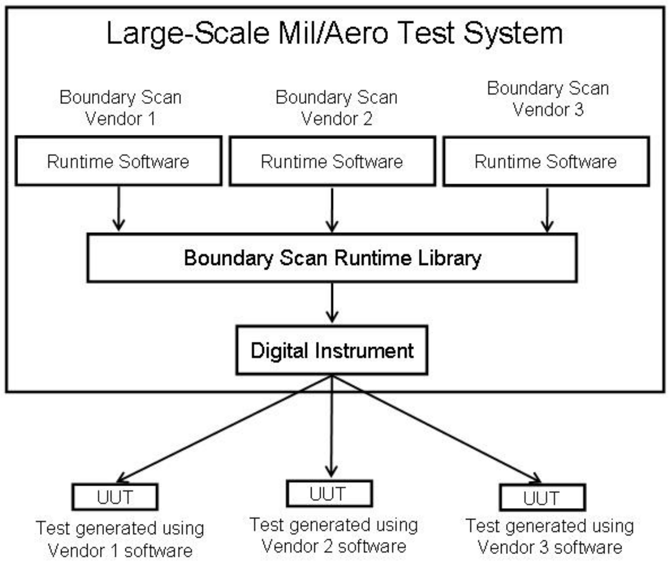
Large-Scale Mil/Aero Test System
Boundary Scan Vendor 1
Boundary Scan Vendor 2
Boundary Scan Vendor 3
Runtime Software
Runtime Software
Runtime Software
Boundary Scan Runtime Library
Digital Instrument
UUT
UUT
UUT
Test generated using Vendor 1 software
Test generated using Vendor 2 software
Test generated using Vendor 3 software

Figure 4. Fixturing with boundary scan runtime library

VII. Desired Attributes

Here are some of the attributes that such a runtime library should aim for:

- The API should be boundary scan oriented.
- The API should be execution-hardware-agnostic.
- The runtime library should be easy for boundary scan vendors to use.
- It should use the underlying test system's hardware resources efficiently.
- It should support parallel I/O in addition to TAP activity.
- It should allow flexible assignment of signals to tester pins.
- It should allow boundary scan vendors to implement a generic execution environment, while still allowing end users to specify things that vary from UUT to UUT, such as pin assignments.

VIII. Proposed Solution

A reasonable approach would be to make a runtime library whose API is at about the same conceptual level as the SVF language [4]. Almost all API elements would be in terms of basic boundary scan concepts. Users of the runtime library (called "clients" below) would have no need for knowledge of the details of operation of the underlying test system.

There would be API function calls to directly specify shifts. TDI data would be fully specified. TDO data could be specified or not, as required by a specific test. The API would support indication that some TDO bit positions are don't cares. There would also be a way to specify parallel I/O, both independent and coordinated with shifting.

The API would support a range of options for determining the results of tests. At one extreme of the range, the client of the runtime library would explicitly fetch raw TDO data after every shift and then perform all the comparisons to determine whether a boundary scan test passed or failed. (Parallel I/O behavior could be handled similarly.)

At the other extreme of result processing, the client would count on the runtime library to do as much result processing as possible. The client of the runtime library would have to specify all expected responses in this case. The runtime library would compare actual behavior against expected behavior itself. The client could then query the overall result, and would proceed to ask for detailed failure information only in those cases where the UUT actually failed. This can be especially efficient on large scale production testers, which typically use digital instruments that include facilities for comparisons against expected values in hardware, so that software need never laboriously examine raw responses of the UUT. Even for failing tests, a client who wants complete failure details could request a list of failing bit positions (or parallel I/O positions) rather than having to infer this himself from raw TDO (or parallel I/O) data.

The runtime library would know the UUT's TAP state at all times. Its API would provide a way to reset the UUT's TAP. It would also provide a way to navigate the UUT to a specific TAP state, or through a specified sequence of states. There would be a way to repeat a stable TAP state a specific number of times. This allows for implementation of burn delays during device programming, and for emulation of the SVF RUNTEST statement.

There would also be ways to specify some test parameters, such as TCK frequency and voltage levels for all UUT signals.

A system using this runtime library would typically test many different UUTs, each of which could potentially connect to a different set of tester pins. To deal with this, the runtime library could accept an "instrument configuration file", by which the end user could specify those details that vary from UUT to UUT. This would allow the TAP port and PIO signals to be arbitrarily mapped onto any set of tester pins (to the extent that the hardware in the underlying tester allows this). This would also provide a place to specify things like unusual voltage levels that only the end user would know.

The runtime library would be designed to take advantage of the appropriate tester features for each testing situation. Individual boundary scan operations requested through the API would be converted into sequences of patterns that would then be executed on the tester. Successive boundary scan operations would usually be packed into a single pattern stream for efficiency, but they could optionally be separated into individual bursts to support interaction after each operation (useful for debugging). Timing would be set up so that parallel IO would be coordinated with boundary scan activity. If a tester supported deep serial memory, then the runtime library would make use of it when appropriate.

Note that use of the runtime library would also insure that robust pin electronics would now be used to apply boundary scan tests. Benefits include protection against dangerous voltages that might otherwise damage the tester, the ability to use a wider range of signal voltages, and the ability to monitor drive states and thus detect UUT faults that prevent the UUT from being driven properly.

IX. Additional Benefits

In addition to the advantages discussed above, creation of a system like this could also streamline development going forward. The creation of a boundary scan runtime library by a digital instrument vendor represents an efficient division of labor between digital instrument companies and boundary scan companies, with each focusing on the aspect of the problem they know best.

The digital instrument vendor has the expertise and is in the best position to optimize performance in a production test environment. If the instrument maker comes up with such an optimization, it benefits each boundary scan vendor but requires no effort by any boundary scan vendor.

Once this API has been defined, the digital instrument vendor could potentially write additional software to support the defined functionality with other hardware, perhaps radically different from the originally-targeted digital instrument. For example, existing high speed subsystem instruments [9] could potentially be configured to support some boundary scan tests. Such a capability could be made straightforwardly accessible simply by supporting the API discussed here. If this were done, then boundary scan vendors who already used that API could run their tests on the new instrument with virtually no additional work on their part.

In fact, if the API for such a system became popular enough, it could potentially evolve into a de facto standard like SVF [4], or an official standard.

X. Summary

Boundary scan has proved its usefulness over the years, so it's not surprising that large scale defense and aerospace test systems are being used to perform more and more boundary scan tests over time. But this increasing prevalence of boundary scan tests has led to fixturing and logistics problems. Each such test is usually developed using the tool set from a specific boundary scan vendor, and a large scale test system must typically execute tests on a variety of UUTs whose tests have been developed using a variety of boundary scan vendors.

Usually, each vendor's tests are developed to run on that vendor's proprietary hardware. Supporting the various hardware from all these vendors in a single test system presents a formidable fixturing and logistical challenge, as has been discussed above. But note that such test systems invariably contain general purpose digital test hardware, which is also capable of executing boundary scan tests. One way to improve this situation would be for the vendor of the general purpose digital test hardware to supply a boundary scan runtime library, which the boundary scan vendors could then use as an alternative way to execute their tests.

References

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

[2] One example is IEEE Standard 1532-2002, “IEEE Standard for In-System Configuration of Programmable Devices”.

[3] XJTAG, “JTAG - A technical overview”, http://www.xjtag.com/support-jtag/jtag-technical-guide.php, accessed July 1, 2011.

[4] ASSET InterTech, “Serial Vector Format (SVF)”, http://www.asset-intertech.com/support/svf.html, accessed July 5, 2011.

[5] EIA/JEDEC Standard JESD71, “Standard Test and Programming Language (STAPL)”.

[6] Fichtenbaum, M. L. and Robinson, G. D. “Scan Test Architectures for Digital Board Testers” IEEE International Test Conference 1990.

[7] Epstein, T. and Allen, S. “Using Deep Serial Memory for Large Block Data Transfers” IEEE AUTOTESTCON 2009.

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

[9] McGoldrick, M. “The Impact of Test Instrumentation with Distributed Processing Capabilities on Test Program Set (TPS) Architecture and Development” IEEE AUTOTESTCON 2011.