Flexible High Performance Architecture For Boundary Scan Execution Hardware

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

***Abstract*—IEEE 1149.1 boundary scan has become very popular since its introduction in the 1990s. It is now used routinely in defense and aerospace testing, and has become commonplace in new TPSes. But mil/aero quality requirements demand functional test as well. This means that boundary scan tests and functional tests must coexist in this environment.**

Boundary scan test requires access to the same UUT I/O signals as functional test. This is because many boundary scan tests, such as interconnect test, require control and observation of these signals to obtain full fault coverage. Functional test requires high-performance dynamic channels on the UUT I/O signals. The boundary scan requirements are less demanding.

This paper proposes the use of configurable test hardware to address both requirements. This is technically achievable now that test systems can make use of FPGAs that can be efficiently reconfigured at run time. This paper proposes a boundary scan test architecture that can be combined with an existing functional test architecture in such a system.

The boundary scan hardware architecture would be based on a specially designed processor optimized for boundary scan which would support up to 8 TAP ports. All available pins not used for TAP ports could be used for parallel I/O signals up to the constraints imposed by the limitations of the FPGA and fixturing. The custom processor could include throughput enhancing features such as pipelined double buffered DMA data transfer. With proper design, such a processor could run continuously as long as it was connected to a host computer that was fast enough to keep it supplied with data. Such a custom processor should be able to closely approach the device limited test time in throughput-critical situations, such as flash programming.

It should be possible to straightforwardly control such hardware using a general purpose boundary scan runtime software library, which would present its client with an API that is conceptually similar to existing boundary scan languages like SVF and STAPL.

Keywords-Boundary scan; JTAG; IEEE 1149.1; Flash programming; Device limited test time; DLTT

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. It is also widely used to program data into programmable devices such as flash memories.

Boundary scan testing coexists with the more traditional functional test techniques that test digital circuitry by exercising it in a manner that approximates the function for which it was designed. Boundary scan can confirm that the unit under test is free of shorts and opens, but it cannot detect subtle defects such as internal component malfunctions that prevent the overall assembly from behaving properly, so mil/aero quality requirements demand functional test as well.

Such functional tests require high-performance dynamic tester channels connected to the UUT I/O signals. In general, boundary scan test requires access to the same UUT I/O signals as functional test. (Some boundary scan testing can be performed by connecting only to a UUT's TAP port, but skipping the connection to the other UUT IO signals sacrifices significant fault coverage.)

Some papers in the past [2, 3] have advocated the use of high performance functional test channels for both types of testing. The initial implementation work on these ideas has been very successful, but note that the early systems of this type simply used the basic capabilities of existing functional test hardware to perform boundary scan tests, and did not go to great lengths to optimize for fast boundary scan execution. This paper describes a way to optimize boundary scan execution throughput in such a system.

The proposal in this paper relies on the fact that digital hardware can now be easily designed in a way that is inherently reconfigurable. This is because digital hardware today is commonly designed using field programmable gate arrays

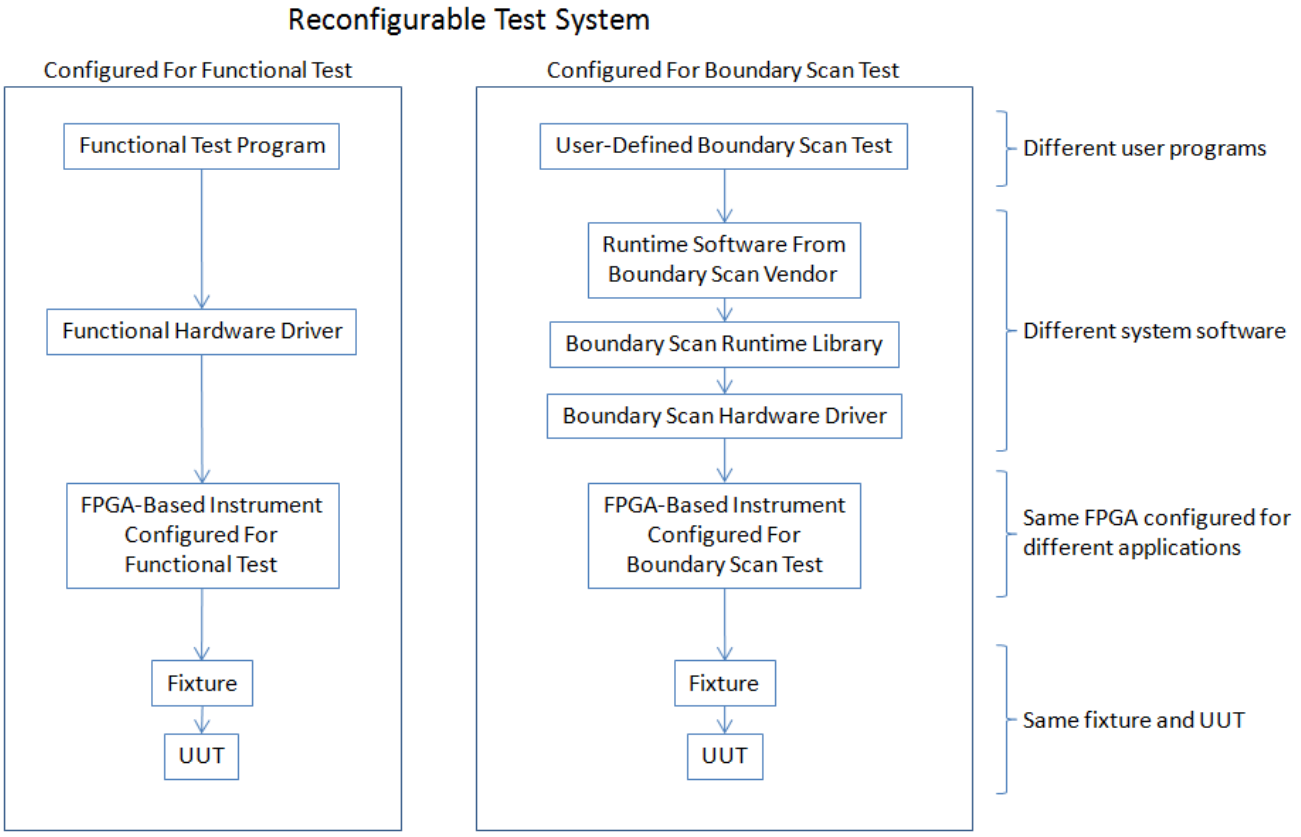

Figure 1. Reconfigurable system for both functional and boundary scan test

(FPGAs). FPGA circuitry is often configured only at the time of manufacture, or infrequently thereafter in order to install field upgrades and bug fixes. But there is no reason why such circuitry can't be designed to allow frequent reprogramming as needed to reconfigure for multiple applications. This technique has been described for such applications as image processing [4], frequency synthesis [5], and reconfigurable computing [6]. This paper proposes that such a technique could be beneficially used to reconfigure a tester to support either high speed functional test or boundary scan test, as shown in figure 1.

Such a system requires an FPGA firmware design optimized for high speed functional testing and an additional firmware design for the same FPGA optimized for fast boundary scan execution. In addition, these FPGA designs must be integrated into a system architecture that provides for very fast reconfiguration of FPGAs. There are systems that can reconfigure their internal FPGAs in times on the order of a second [7]. This paper will not discuss the desired features of a functional tester, but it will discuss what would be desirable for efficient boundary scan execution.

II. Desirable Features for a Boundary Scan Execution Engine

A. Pin Flexibility

When used in production, the intention is to connect the UUT to the tester once, then to run both boundary scan tests and functional tests with no further fixturing or connection changes. To achieve this, the boundary scan engine must connect to the UUT in exactly the same way as the functional tester connects to the UUT. To facilitate this, and to provide maximum flexibility to the test engineer generally, there should be as few constraints as possible on how the channels of the boundary scan engine must be connected to the UUT. The ideal design would allow pins to be assigned to TAP ports however the test engineer wishes, with no further constraints. And ideally, all pins not used for TAP ports should be free to be used for connections to the UUT's parallel pins ("PIOs").

The parametric requirements on the pins are much looser for boundary scan test than for high speed functional testing. Most detailed timing control features (complex drive formats and so on) are not needed, as long as the system can reproduce the relatively simple timing specified by the 1149.1 standard for the TAP port. PIO timing requirements are similarly loose.

However, one timing feature that is desirable is the ability to compensate for the round trip delay over long cables. This mainly means the ability to insert a custom delay in the generation of the strobe time that is used to test each UUT output that is to be tested (TDO for each TAP and any PIOs where tests will be performed).

B. The Need For Speed

Boundary scan was initially designed for simple tests like interconnect testing, and such tests are short enough that optimizing for throughput is not very important. However, once the 1149.1 standard became available, people started using the infrastructure that it provided to program data into programmable parts such as flash memories. Operations like this can sometimes run for tens of minutes or more.

Long flash programming sequences typically occur when the pins of a flash memory to be programmed are accessible only by means of adjacent boundary scan parts. Such a memory can be programmed by repeatedly shifting data through the scan chain to present the same data and control signals to the pins of the flash part that it would see if it were in a standalone programming setup. This will work, but it can be extremely slow, because each increment of data to be presented to the flash requires multiple shifts through the scan chain.

This inherent inefficiency can result in sequences that are tens of minutes long. In analyzing such situations it is useful to calculate the minimum time that the programming sequence would take if it were limited only by the UUT itself (no gaps in waveforms caused by delays in the test equipment that is driving the UUT). In commercial semiconductor test, this minimum time is called the device limited test time (DLTT) [8].

A DLTT analysis for an example flash programming situation would go something like this (with some numbers rounded slightly for brevity).

Consider a flash memory such as the Spansion S70GL02P. Data to be programmed into this device is specified 16 bits at a time. Successive data values must be loaded into a 64 byte write buffer, and whenever the buffer is full, the test system must stop loading data temporarily and command an internal controller within the flash memory to write the recently loaded data into the memory itself. For purposes of this example, let's assume that the design of the UUT requires three shifts for each 16 bit data load. (The "extra" shifts are needed to generate transitions on control lines.) If the scan chain is 1000 bits long, then this example requires 3000 TCK cycles to simply specify each 16 bit piece of data to be programmed into the flash. Let's say that we want to program 10 megabytes of data (5 million 16 bit transfers). That would require 15 billion TCKs (3000 times 5 million) just to specify the data. At a TCK frequency of 10 MHz, that would take 1500 seconds (25 minutes). This does not include the additional time that is spent waiting for the internal operations that save each write buffer into the proper memory locations. The wait time for a write buffer can vary, but the device data sheet gives a typical time of 480 microseconds. We must do about 156250 waits in this case (10 million / 64), and at 480 microseconds per wait this comes to 75 seconds.

C. Hardware and Software Optimized For Speed

This analysis makes it clear that there are real world situations where desired programming tasks could take a very long time even if the maximum speed acceptable to the UUT could be achieved. In such situations, fast execution speed is extremely important. Ideally, one would like to design a system that can come within a few percent of the DLTT. Note that many boundary scan controllers run an order of magnitude or more slower than the DLTT in situations like the example above.

The most important component that is needed for such high speed operation is a special purpose boundary scan processor that executes a stream of instructions specifying the shifts, PIO activity, TAP state navigation, and other operations that make up a boundary scan test. The instruction set for this boundary scan processor should be sufficiently powerful that no intervention by a higher level computer is needed during the entire execution of the flash programming sequence. To achieve this, the boundary scan processor must be able to compare against expected values in hardware, abort the sequence if a failure occurs, and conditionally poll for write buffer completion.

If the boundary scan processor can do all these things on its own, then the programming sequence can be expressed as just a very long stream of instructions, and the rest of the system can simply be an efficient instruction pipeline. If this instruction pipeline can deliver the instruction stream faster than the UUT can accept it, then such a system should be able to run at speeds approaching the DLTT.

Many boundary scan execution systems would have no hope of delivering such an instruction stream at the UUT's DLTT rate because they run software on the system computer to interpretively generate each shift when they are about to execute it. Problems of this type can be overcome by precompiling the required boundary scan activity into an instruction stream ready for execution by the hardware. This compilation process would be slow, but it need only be done once. Another issue with compilation is that the instruction stream could be very large. In the example above, the compiled instruction stream for the data shifts alone would encode data for 15 billion TCKs. There are many options for packing data into such an instruction stream, but even using a rough guess of half a byte per TCK, which seems pretty efficient, would result in 7.5 gigabytes of instruction data. That would be a large amount of data, but note that it should be possible to compress it substantially. Even everyday zip compression should be able to compress such an instruction stream significantly, since there is a huge amount of redundancy in the stream. (In most cases, each shift is identical to the one before it except for a few of the 16 data bits.)

There is also the issue of efficiently delivering this precompiled instruction stream to the boundary scan processor. This could be done with a mechanism like a FIFO, with the boundary scan processor constantly extracting instructions and executing them even as a separate process on the system computer fills the input of the FIFO with new instructions. Another alternative would be for the boundary scan processor to use double buffered instruction memory. While the boundary

scan processor was executing instructions from one of the buffers the system processor could be DMAing the next set of instructions into the other buffer.

A block diagram of the boundary scan execution processor and its instruction pipeline is shown in figure 2.

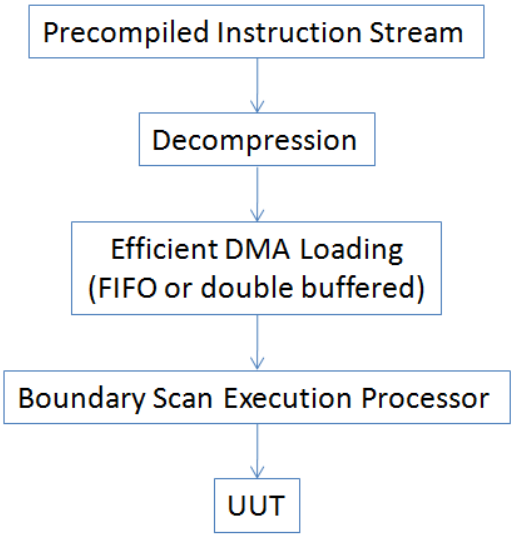

Figure 2. High speed boundary scan execution system

D. *Other Desirable Features*

In addition to the features described above above, it would also be especially good if this boundary scan execution system could gang multiple TAP ports. This would enable even higher flash programming throughput by programming flash memories on multiple identical UUTs in parallel. Designing support for this should be fairly straightforward. The boundary scan activity to be delivered to each of the ganged UUTs would be identical and delivered in lockstep. Special processing would be required to integrate individual responses from the separate UUTs. An example of this would occur during polling loops, when the system would have to do something like wait for all UUTs to be ready before dropping out of the loop and proceeding to the next step in the process.

Support for high TCK frequencies would also be desirable. Short boundary scan tests, such as interconnect tests, are often done at relatively low speeds (in the 1-5 MHz range), but flash programming can benefit from the fastest TCK rate that the UUT can tolerate. For this reason, it would be good for the boundary scan hardware to support TCK rates well in excess of those commonly used. TCK frequencies up to 50 MHz would be a reasonable initial target. Even faster TCK rates would be desirable if UUTs become available that can use them.

E. *Support Using a Standardized API*

Earlier papers have proposed the creation of a standardized API for boundary scan execution using the functional test hardware that is present in large scale Defense and Aerospace test systems [2, 3]. This API, which provides function calls that are conceptually similar to existing boundary scan languages like SVF and STAPL, is now available and is widely supported by boundary scan companies. It should be possible to straightforwardly extend such an API to support the additional capabilities proposed above for efficient operation (compiled instruction streams, polling loops).

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] Borroz, T. "Considerations in the Design of a Boundary Scan Runtime Library" IEEE AUTOTESTCON 2013.

[4] Raikovich, T.; Feher, B., "Application of partial reconfiguration of FPGAs in image processing," in Ph.D. Research in Microelectronics and Electronics (PRIME), 2010 Conference on , vol., no., pp.1-4, 18-21 July 2010.

[5] Tiwari, Abhishek. "A Partial Reconfiguration based Approach for Frequency Synthesis using FPGA" in Procedia Engineering, Volume 30, 2012, Pages 234-241, International Conference on Communication Technology and System Design 2011.

[6] Wikipedia, "Reconfigurable computing", https://en.wikipedia.org/wiki/Reconfigurable_computing, accessed September 2015.

[7] Teradyne, "High Speed Subsystem", http://www.teradyne.com/products/defense-aerospace/high-speed-subsystem, accessed September 2015.

[8] Yost, John, "What Is Concurrent Test?", Evaluation Engineering magazine, April 2009, and also available at http://www2.evaluationengineering.com/enews/200904/0409_design.htm, accessed September 2015.