System Issues in Boundary-Scan Board Test

Kenneth P. Parker
Manufacturing Test Division
Agilent Technologies
Loveland Colorado, USA
Kenneth_Parker@Agilent.com

Abstract

Boards have evolved into complex systems and even collections of interacting systems. Test engineers struggle to find out how these systems are initialized and booted because of poor documentation. While Boundary-Scan (IEEE Std 1149.1) [IEEE93, Park98] is a powerful test tool, test engineers are finding out that yesterday's DFT rules and test approaches may actually be detrimental to successfully testing systems on a board. One culprit is the boot up process of the board and even individual ICs. What can be done to address this?

Introduction

Boundary-Scan is a powerful test tool for complex printed circuit boards and systems, particularly as traditional In-Circuit access through a bed-of-nails becomes increasingly difficult. Today it is not unusual to find a single board with over 500 components (50% digital) and 10,000 nodes with only 2000 (20%) or less actually accessible to In-Circuit probing.

Such boards are extremely difficult to test without a limited access test technology such as Boundary-Scan [IEEE93]. Boundary-Scan adds not only controllability and observability [Will83] but also makes possible the automation of test development and pinpoint diagnostics [Park98]. These benefits are completely lacking in the fallback technology of old, functional testing.

For many years, test engineers have been taught to observe certain testing rules that are absolutely necessary to create rock-solid, repeatable, reliable tests. These rules need to be re-examined now that boards, even individual ICs, may be considered complex system in their own right. Indeed, a board today may consist of many interacting systems. These interactions can introduce new testing problems that can be especially baffling for those who are unprepared.

Features of Board Systems

Twenty years ago, boards were composed of medium-scale integrated circuits and only worked as a system with other boards. This meant that a single board probably did not have enough logic on it to constitute a system. The board often did not have an oscillator either. Thus, the board did not "boot up" but more likely responded to a master reset signal. This reset signal was a natural target for test engineers, because they knew that asserting the reset would instantly bring the board into a well-defined state.

But a few boards did have an oscillator and this meant that after power was applied, the board was probably trying to "do something". In these cases, test engineers were quick to develop the rule that as well as asserting the master reset, they should also "kill the clock" by disabling it. Disabling the clock had three salutary effects. First, it prevented the board (which was likely only a portion of a system) from trying to do something that, due to the incompleteness of the system, was impossible. Second, it decreased the likelihood that the board might stumble into an inconsistent state (where, for example, a bus-fight might result) and third, it cut down on electrical noise that might complicate testing. Most systems of the past only had a single oscillator.

Today and as far out as we can see, a board is a complex system, or even a collection of interacting systems. A board may contain hundreds of ICs, some with 10^5 to 10^6 gates, and contain multiple clock domains. Each domain could be a system in its own right, with asynchronous communication protocols between them. Even more

interesting is that fact that these "clocks" may be hidden from the view of the test engineer because they are buried within an IC. The board designer may soon forget these "crypto-clock" domains are there because they are not a part of the mission function of the board, but part of the mission infrastructure.

A perfect example of crypto-clock domain infrastructure is illustrated by a hypothetical[1] Field-Programmable Gate Array (FPGA) paired with a complementary serial PROM (SPROM) as shown in Figure 1

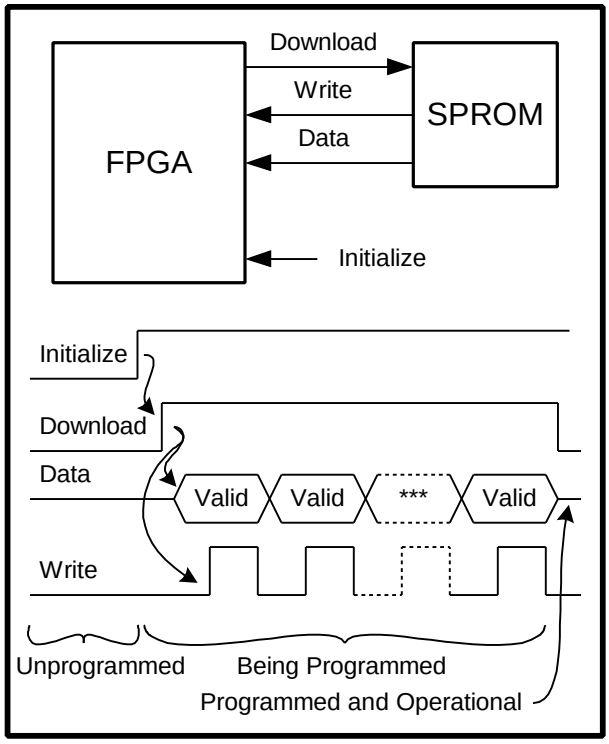

Figure 1: FPGA/SPROM Configuration

Upon receiving an initialization signal (the rising edge on Initialize), an FPGA may request a download from an SPROM (the rising edge on Download) which then communicates via a serial protocol with the FPGA, under the control of an on-chip clock buried in the SPROM (using the Write and Data signals). This download of mission functionality takes place in a few hundred milliseconds after the request and then the FPGA takes on its mission function. Indeed, the Initialize signal may be tied to the positive voltage supply such that the act of applying power to the board triggers the FPGA configuration as part of the power-up process. Again the board designer probably never counts this activity as a new clock domain. Worse, the test engineer, who cannot hope to be an expert in all ICs[2] that will pass his or her eyes, may not be aware that the autonomous serial protocol even exists.

Figure 2 shows a second common scenario, where a microprocessor has the capability of programming an FPGA directly. This is an attractive design because it allows a circuit to be dynamically reconfigured under the control of software.

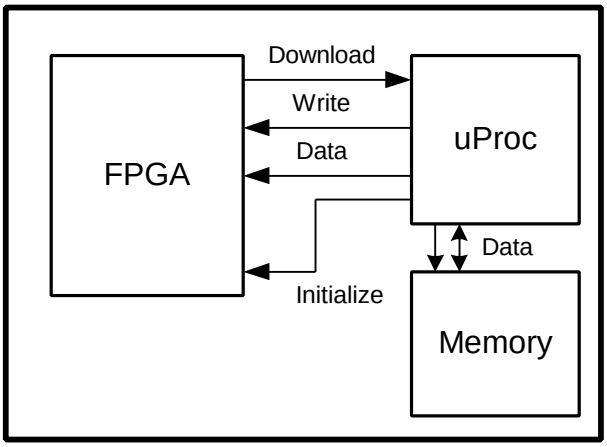

Figure 2: An FPGA directly programmed by a microprocessor, allowing dynamic reconfiguration.

Some Testing Problems of Systems

What does the particular problem just described have to do with Boundary-Scan testing? Perhaps not too much *if* the serial download never interferes with the operation of Boundary-Scan. Sadly, it may well have serious consequences when it does. There are at least two scenarios known where trouble results. In one scenario, Boundary-Scan in the FPGA simply doesn't work while the serial download is operating. In another, the FGPA is incapable of communicating with its neighbors before or during the download because the device has pin parameters (like V_{IH} and V_{OL}) that are determined *by the download*. Thus, if the board

[1] This example is simplified but representative of actual devices.

[2] Particularly when some of the ICs contain programmable logic that is undocumented.

does not successfully program such devices by itself, then Boundary-Scan will not operate correctly until they are programmed. The board test engineer may not be (at first) aware of this requirement and it may be a difficult requirement to satisfy if proper consideration is not taken.

If a board is a system, then we must view its test as a system test problem. We must be particularly wary of that critical time from the point of power application[1] to when the board becomes "operational". This is where many test problems are born.

An Example

A board being tested was a collection of four obvious systems, each containing a microprocessor. Each microprocessor had a nearby FPGA it could dynamically load with programming. In addition, there were a dozen additional FPGAs that loaded their programming from nearby SPROMs. An example of the problem with autonomous FPGA programming is illustrated in Figure 3.

This board powers up and then downloads programming bits into its FPGAs. Some of these FPGAs are triggered by power up directly, by sensing a rising edge on the power supply rails. Others are triggered by signals ultimately derived from the microprocessors. When testing begins, "foreign" signal activities occur which can trigger a programming initiation pulse on an FPGA. This causes the FPGA to begin its autonomous programming sequence, except now the test prevents it from completing, since the system has been "lobotomized" [Park98] by Boundary-Scan, that is, it no longer can respond as the designer intended for program downloading. Unfortunately, these FPGAs are not compliant during the download process. In this real case, certain FPGAs became non-compliant *during* Boundary-Scan testing, ruining the integrity of the Boundary-Scan infrastructure and leading to nonsensical failure diagnostics. In this case, the only way to recover the proper operation of the board was to cycle the power.

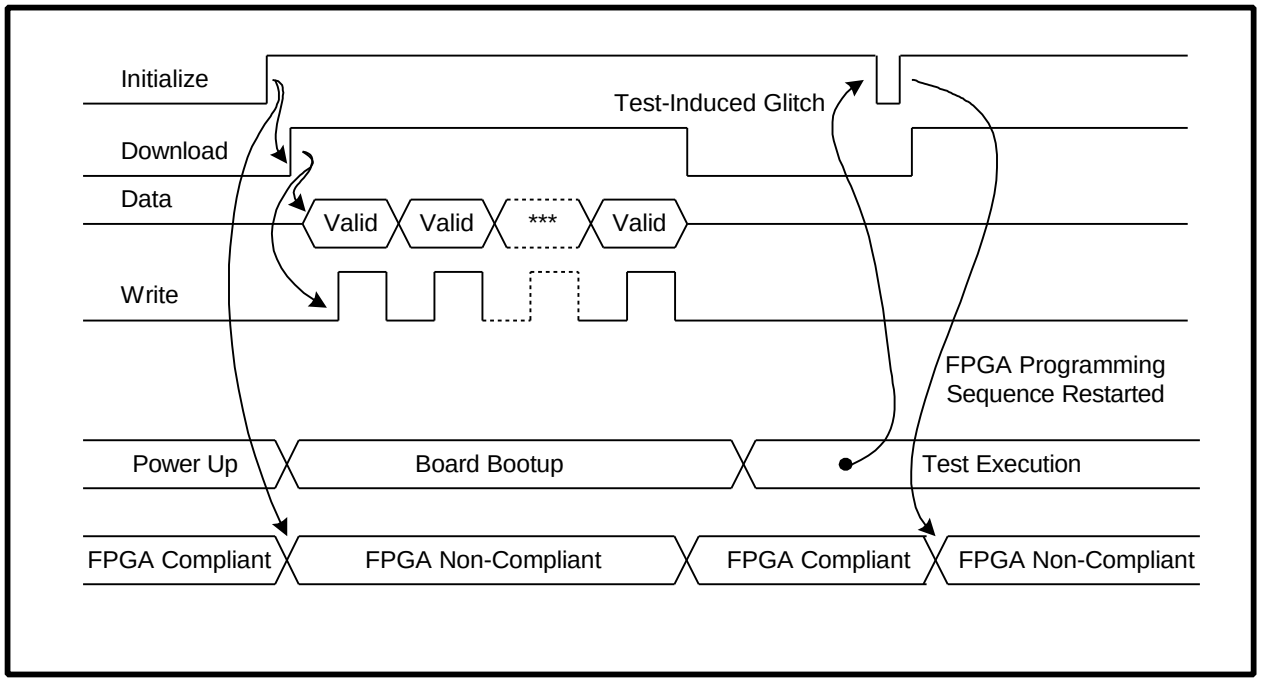

Figure 3: A sequence of events where a Boundary-Scan test is invalidated because it causes an FPGA to become non-compliant during testing.

This board was especially troublesome since at the time of test initiation, it was not clear if *all* the FPGAs were in a compliant state. It is also difficult to determine if the test itself would cause any FPGAs to suddenly begin a programming sequence.

A board designer may be forgiven for not

[1] We simplify this problem here by assuming only one power supply where several may exist, that could be turned on in sequence.

having a "failing" mindset. His/her job is hard enough just trying to fathom the correct operation of the design without considering all the way it might not work. Thus, the natural inclination of designers is to overlook the issue of what might happen during the power-up sequence that could frustrate its success.

However, the test engineer must consider how a board powers up, and what could go wrong with that process. The next sections discuss old and new ways of looking at this problem.

Some Previous DFT Rules

In the past, test engineers wanted to control a board from the very beginning, as soon as power was applied. So, two rules have resulted: 1) Assert any reset you can find; and 2) kill any oscillators you can find, preferably with the ability to insert a tester controlled clock signal[1] at will. These two rules guaranteed that a board would, at power up, be quiescent and very likely in a consistent state (i.e., with no drivers in conflicting states). From there it could (usually) be safely and repeatably In-Circuit tested.

In boards common today, there may not be an easily identified reset, and there may be several (even many) clock domains. Further, one may not find all the clock domains simply by looking at the bill-of-materials for oscillator devices because of crypto-clock domains created by buried oscillators. Finally, since many boards now contain configurable logic devices, the configuration of the board may not be established until significant time has passed after power up, *provided nothing interferes with the configuration process*.

New DFT Considerations

Today's test engineer is faced with new considerations. Should master resets be asserted during power up? Should master oscillators be shut down? The answer seems to be "maybe". It is very helpful to have the board designer available to discuss these questions, but that is often an unobtainable luxury. Thus we need new DFT rules which can be used to frame this discussion early in the design.

There seems to be two approaches to powering up for test. One, the old approach, is to disable all clocks and assert all resets continuously as power up occurs. We call this an "unbooted board". The other, newer approach is to allow the board to boot itself up after power application by waiting for some interval of time beyond power stabilization for the intelligent actors (microprocessors executing PROM boot code, FPGAs conducting autonomous downloads, etc.) to complete their boot processes. This is called a "booted board". If testing a board depends on it being successfully booted before a test technology such as Boundary-Scan will work, then we have new considerations. Here are some questions regarding the testability of a board:

1. Does the unbooted board have a working configuration, or does it depend on the boot process[2] to establish the ability for ICs to logically communicate?
2. How sensitive is a booted board's boot process to disruption? For example, would a commonly expected failure prevent the board from booting up successfully? This is akin to judging the fraction of possible failures that can prevent boot up and asking if the risk is acceptable.
3. If boot up is critical to success, how can we tell if we have achieved it? This may prevent a lot of "untestable" boards ending up in a bone pile.
4. If boot up is critical to success, which nodes should *not* be probed with In-Circuit nails because of loading effects that can prevent the boot process from operating correctly? For example, nails on system clocks should be avoided to ensure the quality of these critical signals.
5. Are there processes on the board that, once triggered, prevent testing activities from working? The autonomous FPGA download is an example of this problem.
6. If such a process can be triggered, how can we avoid it? If triggered, will it complete? How can

[1] This tester-controlled clock usually had a much lower clock frequency and would only be injected during the testing of certain devices that needed it.

[2] Here we mean a full boot up, or enough boot up to assure that testing mechanisms are successfully enabled.

we tell if it completed? If it won't complete, what mechanism[1] may be used to reset it?

Discussing these questions can quickly lead to design and test practices (rules) that can avoid many problems during volume production. These problems will usually lead to poor yield at board test and bad boards that are very difficult to diagnose, so there is a big payoff in heading them off.

Conclusion

Modern boards, now collections of systems, some of which are easily spotted, and others which are hidden (called crypto-clock domains) present new problems for test engineers and the teams that design them. It is especially useful to form a test strategy early in the design that decides whether a board must be unbooted or booted before testing activities begin.

[1] As noted in the example, some FPGAs refuse to perform Boundary-Scan while being programmed, and this process may never complete if the expected serial process is disrupted. Indeed the only way to recover may be to cycle power on the FPGA.

REFERENCES

[IEEE93] "IEEE Standard Test Access Port and Boundary-Scan Architecture", IEEE Standard 1149.1a-1993, *IEEE Standards Board*, 345 East 47th St. New York, NY 10017, 1993

[Park98] K. P. Parker, "The Boundary-Scan Handbook, 2nd Edition, Analog and Digital", Kluwer Academic Publishers, Norwell MA, 1998

[Will83] T. W. Williams and K. P. Parker, "Design for Testability – A Survey", *Proceedings of the IEEE*, vol. 71, No. 1, Jan 1983