A book on system test, and testing systems also

Scott Davidson
Sun Microsystems

When I saw System-level Test and Validation of Hardware/Software Systems at last year’s ITC, I was excited. I’ve been fascinated by system test ever since I discovered that more people in my company were working on it than were working on IC test, but I knew of no good books on the subject. I discovered that only part of this book covered the kind of systems, such as servers, that I was thinking of. Some chapters cover systems as in SoCs. And some cover both types, from the viewpoint that system level refers to register transfer level and above. This book is the result of a European-Union-funded project: COTEST, Testability Support in a Co-design Environment. Composed of reports from several different research projects, the book’s overall structure is excellent, with nine well-integrated chapters.

A major impediment to advancing the utility of high-level system test is the lack of a widely accepted fault model. I was happy to see that the book begins with the best survey (and perhaps the first) of high-level fault models that I know of. Most of these models have been derived from software test coverage metrics. Of the papers I’ve read in this area, most are flawed by a high-level coverage metric that’s too optimistic for any reasonable design. Chapter 2 explains why this is so, and why the observability problem in hardware makes these metrics less effective for our applications. The problem is still not solved, but I’d strongly recommend anyone beginning work in this area to read and understand this survey. Many of the chapter’s ideas could readily be applied to measure the test coverage of “real” systems—systems that are shipped to the end customer.

Chapters 3 through 6 cover high-level test generation, starting with circuit descriptions written in a hardware description language. Chapter 3 presents a symbolic approach, mixing random test generation using genetic algorithms—with deterministic test generation, using binary decision diagrams (BDDs.) Results are measured using a bit coverage model, with faults injected into the design, much as in traditional fault simulation. Results are given for a set of reasonably small benchmarks (fewer than 10,000 gates). Coverage is good when the methods are combined, but test generation times are long. I’d prefer data from larger designs, to indicate if the technique would be useful for production circuits.

The next test generation method is heuristic, and starts with a behavioral-level model. Two fault models are discussed in this chapter, one bit oriented, and the other based on the condition of an if statement. Coverage information is achieved by injecting the faults and performing serial fault simulation. Tests are generated by mutating an existing set of vectors. This technique is quite useful for functional test, since design verification vectors already exist and we are primarily interested in improving their coverage. Results were measured on a set of small benchmarks, and compared against a rather antiquated sequential ATPG.

Most attempts at hierarchical test generation involve inserting faults in blocks modeled at the gate level, and propagating and justifying the test at a
higher level. Hierarchical test generation as proposed in Chapter 5 is a bit different—it makes use of decision diagrams (DDs) based on structural information to assist in test generation at the behavioral level. A whole circuit can be split into control and data path DDs at the high level, and each of these is composed of a set of lower-level DDs. Structural models are often simpler to use than behavioral models, because specific paths through logic blocks can be found. So, although in principle the work described in this chapter is similar to traditional hierarchical ATPG, it has the advantage of being built around a more integrated modeling philosophy. Results on one small benchmark are given, but no test generation times, so the technique’s practicality is not immediately clear.

Chapter 6 covers test program generation from high-level microprocessor descriptions. While the method produces good results, the reason I think everyone working in high-level test generation should read this particular chapter is because it recognizes that no top-level method can ever achieve the best fault coverage as measured at lower levels. The method described here incrementally generates tests as the design is refined. A highlight is Table 6.6, which qualitatively compares coverage metrics, showing how a test optimized to detect faults under one metric performs on others. For instance, a test that achieves 92.35% branch coverage reaches 99.04% statement coverage, while a test targeted to statement coverage, and achieving 98.49% under that metric, reaches only 85.49% branch coverage. The authors of this chapter have thought through the important issues in high-level test generation, with perhaps the most illuminating results I have ever seen.

Chapter 7 covers real system test. One set of faults that traditional techniques do not model is synchronization errors. After presenting a useful refresher course on synchronization, the authors introduce a synchronization fault model. The model has two classes: static faults, which have a wrong value that can be observed at any time, and timing faults, where the value may be correct but the time at which it is observed is incorrect. The chapter covers only the detection of these faults, not test generation, and some interesting results are given. The chapter is very clear but only 13 pages long; I found myself wanting more discussion on other timing faults.

"An Approach to System-level Design for Test," Chapter 8 sounds as though it is very high level; unfortunately, it's not. The chapter describes a way of trading off pseudorandom test against deterministic test, using stored patterns. Given a memory constraint (either tester or internal pattern storage), test time is minimized. The system discussed in this chapter is a SoC, not an end-user system. I am not sure where this test is supposed to be applied. Constraints of tester pattern memory are discussed, but this is unlikely to be a problem when known compression techniques are used. Test time is also not likely to be a problem, since BIST is not very constrained by ATE speeds, and is thus unlikely to be a test time limiter. In the field, this is even less likely to be the case, and in my experience even when deterministic top-up vectors are used in manufacturing test, a purely random test is adequate for the field, where very high fault coverage is not necessary. Finally, in some cases like this one, minimization is not necessary, and simpler heuristics than the procedures described in this chapter are sufficient.

The final chapter is again about end-user systems. These systems have a greater variety of things that can go wrong than ICs, and they can be repaired. Fault trees model how failure events can cause the failure of subsystems or of complete systems. Dynamic fault trees allow the modeling of dependence between events, can model events that are forced to occur in a given order, and can also model the effect of spares. Modeling sparing also allows the modeling of the repair process. After this background, the authors present an example of a reliability analysis. The chapter covers a lot of useful ground, is clearly written, and has a meaningful example.

I wish this book had more on real systems, but I'm probably in the minority in that regard. The chapters' average quality is very high, and the ones on high-level fault models and test generation especially have already been useful to me. Don’t expect to find the answers to the problem of effective system test here. That’s intractable, but this book moves the conversation along very nicely. If you are working or thinking of working in this area, System-level Test and Validation of Hardware/Software Systems might save you from going down some dead ends.

Direct questions and comments about this department to Scott Davidson, Sun Microsystems, M/S USUN07-107, 324 N. Mary Ave., Central Research Park, Sunnyvale, CA 94085; scott.davidson@sun.com.

For further information on this or any other computing topic, visit our Digital Library at http://www.computer.org/publications/dlib.