

Embedded Software-Based Self-Test for Programmable Core-Based Designs

Angela Krstic

University of California, Santa Barbara

Wei-Cheng Lai and Kwang-Ting Cheng

University of California, Santa Barbara

Li Chen

University of California, San Diego

Sujit Dey

University of California, San Diego

The programmable cores on SoCs can perform on-chip test generation, measurement, response analysis, and even diagnosis. This software-based approach to self-testing enables at-speed testing and incurs low DFT overhead.

■ **WITH THE GROWING** popularity of system-on-a-chip (SoC) architectures, demands for short time to market and rich functionality have driven design houses to adopt a new core-based SoC design flow. A core-based SoC incorporates multiple complex, heterogeneous components on a single piece of silicon; these can include digital, analog, mixed-signal, RF, micromechanical, and other kinds of systems. This blurring of the boundaries between different types of devices, together with rapidly increasing operational frequencies and shrinking feature sizes, has introduced a whole new set of testing challenges.

Not only are high-speed testers costly, but also their performance is increasing more slowly than device speed. Thus, externally testing SoCs translates into increasing yield loss, because guardbanding to cover tester errors results in the loss of increasingly more good chips. Because digital logic testers cannot do precise analog testing, externally testing mixed-

signal chips requires an even more expensive mixed-signal tester and a two-pass strategy. For some designs, testing the digital core involves pumping a vast amount of digital data through an analog interface; for these, neither of the two platforms would work.

Built-in self-test (BIST) solutions eliminate the need for high-speed testers and can more accurately apply and analyze at-speed test signals on chip. Existing *structural BIST* techniques, such as scan-based BIST, offer good test quality but require additional dedicated test circuitry, so they incur nontrivial area, performance, and design time overhead. Moreover, structural BIST's nonfunctional, high-switching random patterns consume much more power than normal system operation. Finally, to apply at-speed tests to detect timing-related faults, existing structural BIST must resolve various complex timing issues related to multiple clock domains, multiple frequencies, and test clock skews that are unique in test mode.

A new paradigm, *embedded software-based self-testing*, could alleviate the problems of both external testers and structural BIST.¹⁻³ In this strategy, sometimes called functional self-testing, the SoC's programmable cores first undergo self-test by running an automatically synthesized test program that achieves high fault coverage. Next, the programmable core functions as a pattern generator and response analyzer to test on-chip buses, interfaces

between cores, and even other cores, including digital, mixed-signal, and analog components. In addition, functional information can guide diagnostic self-test program synthesis.⁴

Here, we give an overview of the existing embedded software-based self-testing and self-diagnosis methods for core-based SoC designs, and we discuss the challenges to further developing this new testing paradigm.

Concept and advantages

Figure 1 illustrates the embedded software-based self-testing concept, using a bus-based SoC as an example. The SoC's IP cores connect to a peripheral component interconnect (PCI) bus via the virtual component interface (VCI), which acts as a standard communication interface. First, the microprocessor tests itself by executing a set of instructions. Next, it tests the bus and the other nonprogrammable IP cores. To support the self-testing methodology, the IP core lies inside a test wrapper containing test support logic to control scan chain shifting, buffers to store scan data and support at-speed test, and so on. In this example, the on-chip bus is a shared bus to which the arbiter controls access.

The embedded software-based self-test approach has several advantages. First, because this strategy reuses the SoC's programmable components for testing, it minimizes the addition of dedicated test circuitry for DFT or self-test. Second, besides avoiding the cost of high-speed testers, this technique reduces the yield loss stemming from tester accuracy problems. It can also apply and analyze at-speed test signals on chip more accurately than testers can. Third, whereas hardware-based self-test must take place in the nonfunctional BIST mode, software-based self-test can proceed in the design's normal operational mode—the core applies tests by executing instruction sequences as in regular system operations. This eliminates the excessive power consumption of hardware BIST. In addition, functional self-test avoids the overtesting

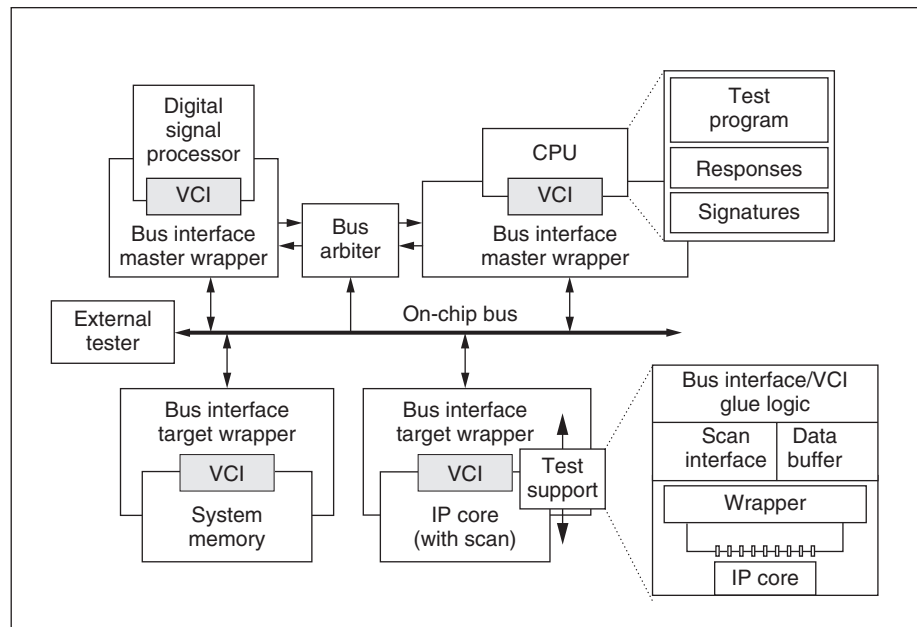


Figure 1. Embedded software-based self-testing concept.

that arises from the application of nonfunctional patterns during structural delay testing (through at-speed scan or BIST). Experiments have shown that many structurally testable delay faults in microprocessors can never be sensitized in the circuit's functional mode.² Thus, defects on these faults do not affect circuit performance. However, testing by applying nonfunctional patterns could detect these defects and unnecessarily identify a chip as faulty. Finally, this methodology suggests DSP-based approaches that offer a promising alternative to the costly process of functionally testing analog circuits.⁵

Embedded processor self-testing

Researchers have proposed several self-testing approaches for microprocessors (see the "Microprocessor self-test" sidebar, next page). Unlike hardware-based self-testing, software-based testing is nonintrusive; it applies tests in the circuit's normal operational mode. Moreover, software instructions can guide the test patterns through a complex processor, avoiding test data blockage arising from nonfunctional control signals, as occurs with hardware-based logic BIST.

Our embedded software-based self-test methods include two steps: test preparation and self-testing.¹⁻³ Test preparation involves generating what we call *realizable* tests for the

Microprocessor self-test

The literature of testing includes several proposals of microprocessor self-testing approaches. Brahme's and Abraham's approach, involving functional testing without any knowledge of the processor's structure, resulted in low fault coverage.¹ Techniques exist for efficiently compiling self-test programs for embedded processors, but they do not address self-test program generation.^{2,3}

Some researchers have advocated applying randomized instructions to the processor under test.^{4,5} However, although processors are more amenable to random-instruction tests than to random-pattern tests, it is difficult to target structural faults by applying random instructions at the processor level. Lee and Patel described an approach that uses structural ATPG to generate tests for stuck-at faults in the processor.⁶

All these approaches target only stuck-at faults, and the methods cannot be easily generalized for delay faults.

References

1. D. Brahme and J.A. Abraham, "Functional Testing of Microprocessors," *IEEE Trans. Computers*, vol. 33, no. 6, June 1984, pp. 475-484.
2. G. Kruger, "A Tool for Hierarchical Test Generation," *Trans. CAD*, vol. 10, no. 4, Apr. 1991, pp. 519-524.
3. U. Bieker and P. Marwedel, "Retargetable Self-Test Program Generation Using Constraint Logic Programming," *Proc. Design Automation Conf. (DAC 95)*, ACM Press, New York, 1995, pp. 605-611.
4. J. Shen and J.A. Abraham, "Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation," *Proc. Int'l Test Conf. (ITC 98)*, IEEE Press, Piscataway, N.J., 1998, pp. 990-999.
5. K. Batcher and C. Papachristou, "Instruction Randomization Self Test for Processor Cores," *Proc. VLSI Test Symp. (VTS 99)*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 34-40.
6. J. Lee and J.H. Patel, "Hierarchical Test Generation under Architectural Level Functional Constraints," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, Sept. 1996, pp. 1144-1151.

processor's components—these are tests that can be delivered through instructions. The processor executes a special software program and stores the results in memory. Analyzing the results can tell us if there are defects in the processor. To avoid producing undeliverable test patterns, our methods generate the tests under the constraints of the processor instruction set. The tests can then be either stored or generated on chip, depending on which method is more efficient for the particular case.

If the tests are generated on chip, a self-test signature characterizes each component's test

needs. This signature includes seed S and configuration C of a pseudorandom number generator as well as the number of test patterns to be generated, N . A pseudorandom number generation program can expand the self-test signatures on chip, into test sets. A component can have multiple self-test signatures, if necessary. Thus, our self-test methodology lets us incorporate any deterministic BIST techniques that encode a deterministic test set as several pseudorandom test sets. A low-speed tester can load the self-test signatures or the predetermined tests to the processor memory before test application.

In the self-testing step, shown in Figure 2, a software tester applies the realizable tests. It also compresses the responses into self-test signatures that are then stored in memory. An external tester can later unload and analyze the signatures. (We assume that before test application, the processor memory has been tested with standard techniques such as memory BIST and is free of faults.)

Testing stuck-at faults

In our work, we have proposed a software-based self-test method that targets structural faults in a processor core using a divide-and-conquer approach.¹ First, it determines the structural test needs for the processor's subcomponents, such as the ALU and program counter. Next, the component tests are either stored or generated on chip. Then, the processor delivers the tests to their target components using predetermined instruction sequences.

To ensure that the test patterns generated for a particular subcomponent are deliverable by instructions, we first derive the instruction-imposed constraints for each component. We can divide these constraints into input and output classes: Input constraints define what input space the instructions allow for the component. Expressed as Boolean equations, they describe the correlation among the component's inputs. Output constraints define the subset of component outputs observable by instructions. In addition, we can classify the processor's instruction set constraints as either spatial (specifiable in a single time frame) or temporal (spanning several time frames).

If we use automatic test-pattern generation to generate component tests, we can specify the spatial constraints during test generation with the aid of the ATPG tool. On the other hand, if we use random tests for components, we can use random patterns only on independent inputs. We use component-level fault simulation to evaluate these tests' preliminary fault coverage. We can evaluate the final fault coverage with processor-level fault simulation once we have constructed the entire self-test program.

After we have derived the realizable component tests, the next step is on-chip self-test using an embedded software tester for test generation (if desired), test application, and test response analysis. Because we have developed the component tests under the instruction set constraints, it will always be possible to find instructions for applying the component tests. On the output end, we must take special care when collecting component test response. Data outputs and status outputs have different observability levels and should be treated differently during response collection. In general, although there are no instructions for storing a component's status outputs directly to memory, we can use conditional instructions to create the image of the status outputs in memory. We can use this technique to observe any component's status outputs.

Using manually extracted constraints, we have applied this self-test scheme to a simple Parwan processor. The method generated a high-coverage test program for the simple processor, demonstrating the feasibility and effectiveness of software-based self-test.

Delay testing

We have also proposed a software-based self-test method aimed at delay faults in processor cores.^{2,3} Given the instruction set architecture and the processor core's micro-architecture, we first extract the spatial and temporal constraints between and at the registers and control signals. Next, a path classification algorithm implicitly enumerates and examines all paths and path segments. If a path cannot be sensitized with the imposed extracted constraints, the path is functionally untestable and, thus, eliminated from the fault

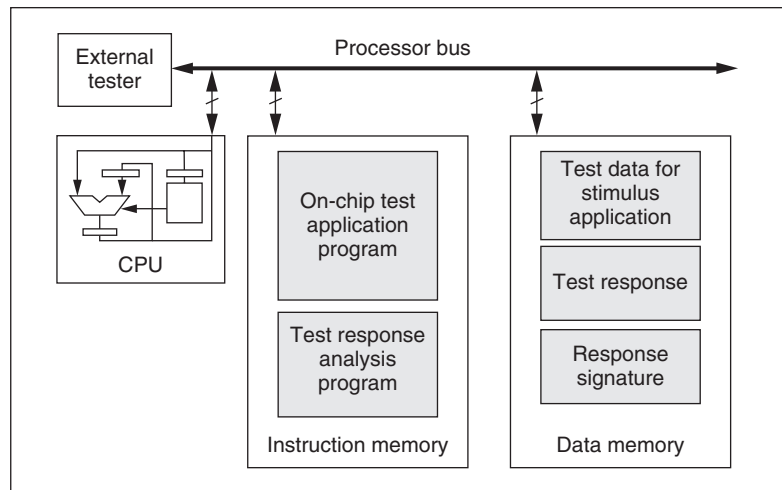


Figure 2. Embedded processor self-testing.

universe. This helps reduce the computational effort of the subsequent test generation process. Experimental results show that a high percentage of the paths are functionally untestable.²

Next, we select a subset of long paths among the functionally testable paths as targets for test generation. We extend a gate-level ATPG for path delay faults to incorporate the extracted constraints into the test generation process, and this ATPG generates test vectors for each target path delay fault. If the test is successfully generated, it not only sensitizes the path but also meets the extracted constraints. Therefore, it is most likely deliverable by instructions (if we can extract the complete set of constraints, we can guarantee delivery by instructions).

In the test program synthesis process that follows, the test vectors specifying the bit values at internal flip-flops are first mapped back to word-level values in registers and values at control signals. These mapped value requirements are then justified at the instruction level. Finally, we use a predefined propagating routine to propagate the fault effects captured in the registers and flip-flops of the path delay fault to the memory. This routine compresses the contents of some or all registers in the processor, generates a signature, and stores it in memory.

We then repeat the procedure until we have processed all target faults. We generate the test program offline and then use it to test the microprocessor at speed.

To apply the synthesized test program, we use an external low-speed tester to load it into

the processor's on-chip memory. Depending on the processor core's architecture, the external tester might need to reset the processor or apply an initialization sequence to make the processor begin fetching and executing instructions from memory. When the test program runs at speed, the processor records a set of signatures in memory. At the end of the test, the test program calls a response analysis subroutine to further compress the recorded signatures in memory and, finally, to compare the compressed signature with the correct signature.

We have applied this test synthesis program to Parwan and DLX processors. On average, for the Parwan processor our method took 5.3 instructions to deliver a test vector and achieved 99.8% fault coverage for testable path delay faults. For the DLX processor, our method took 5.9 instructions to deliver a test vector and achieved 96.3% fault coverage for testable path delay faults.

Embedded processor self-diagnosis

One benefit of software-based self-test is that we can apply it without scan chains, making it a suitable choice for designs that cannot tolerate scan-induced performance overhead on their critical paths, such as high-end microprocessors. The absence of scan chains, however, poses a significant challenge for fault diagnosis.

Recently, researchers have proposed several methods to generate diagnostic tests for sequential circuits by modifying existing detection tests. A prerequisite for these methods is a high-coverage detection test set for the sequential circuit under test; we cannot diagnose a fault if we cannot detect it. Unlike sequential ATPG techniques, software-based self-test has the potential to successfully generate tests for a particular type of sequential circuit—microprocessors. If properly modified, these tests might achieve high diagnostic capability. In addition, software-based self-test has great potential in diagnosis, because functional information is an invaluable resource for guiding and facilitating diagnostic test generation.

To use software-based self-test for diagnosing stuck-at faults in microprocessors, we analyze the combination of test responses to many fine-grained diagnostic test programs.⁴ For high diagnostic resolution, we generate the diag-

nostic test programs so that each test program detects as few faults as possible but the union of all test programs detects as many faults as possible. Experimental results on the Parwan processor example show promising results. With a reasonable number of diagnostic test programs, we can achieve high diagnostic resolution on an overwhelming majority of faults in the processor (data path faults and most faults in the control logic).

Currently, this method requires long fault simulation time for evaluating the diagnostic test programs. However, we found that many of the test programs we used in our study were redundant because of correlation among them.⁴ Thus, a more deterministic diagnostic test generation method could greatly shorten the fault simulation time.

Self-testing buses and global interconnects

In SoC designs, many core-to-core communications require long interconnects. As gate delays continue to decrease, overall performance depends increasingly on interconnect performance. However, because of the increase of cross-coupling capacitance and mutual inductance, signals on neighboring wires can interfere with one another, causing excessive delay or signal integrity loss. Because crosstalk relates to timing, testing for its effects must take place at the circuit's operational speed.

A software-based methodology can address the problem of testing system-level interconnects in embedded processor-based SoCs, the most dominant type of SoC.^{6,7} In such SoCs, the embedded processor cores can access most of the system-level interconnects, such as the on-chip buses. In our methodology, an embedded processor core in the SoC executes a software program that tests for crosstalk effects in these interconnects. The self-test program applies the test vector pairs to the appropriate bus in the system's normal functional mode. In the presence of crosstalk-induced glitch or delay effects, the vector pair's second vector becomes distorted at the bus's receiver end. The processor stores this error effect to the memory as a test response, which an external tester later unloads for off-chip analysis.

In a core-based SoC, the address, data, and control buses are the main types of global interconnects with which the embedded processors communicate with memory and other cores via memory-mapped I/O. Chen and colleagues concentrate on testing the data and address bus in a processor-based SoC.⁶ Their approach uses the maximum aggressor (MA) fault model to model crosstalk effects on the interconnects.⁸

The MA fault model abstracts the crosstalk defects on global interconnects by a linear number of faults. It defines faults based on the resulting crosstalk error effects, including positive glitch (g_p), negative glitch (g_n), rising delay (g_r), and falling delay (g_f). For a set of N interconnects, the MA fault model considers the collective aggressor effects on a given victim line Y_i , while all the other $N-1$ wires act as aggressors. The patterns, called MA tests, are derived such that they excite the worst-case crosstalk effects on the victim line.⁸ For a set of N interconnects, there are $4N$ MA faults, requiring $4N$ MA tests. These $4N$ faults cover all physical defects and process variations that can lead to any cross-coupling-induced crosstalk error effect on any of the N interconnects.⁸

Even though the MA tests can cover all physical defects related to crosstalk between interconnects, many of these defects can never occur during normal system operation, because of system-imposed constraints.⁷ Therefore, testing buses using MA tests might screen out chips that are functionally correct for any pattern produced under normal system operation. As an alternative, we propose functionally maximal aggressor (FMA) tests, which meet the system constraints and are deliverable under the functional mode.⁷ These tests completely cover all crosstalk-induced logical and delay faults that can cause errors during the functional mode.

Given the timing diagrams of all bus operations, we can extract the spatial and temporal constraints imposed on the buses by the functionality of the bus protocol or by the processor core. Next comes FMA test generation. These tests represent vectors that are applicable in the functional mode and that enable the maximal number of aggressors to a victim wire. The strategy uses a covering relationship between

vectors extracted from the bus commands' timing diagrams during FMA test generation.

We could map each FMA test directly into an instruction sequence. However, the simple mapping approach might lead to a lengthy program that would dramatically increase test application time. To reduce the test program's length, we compact the FMA tests before translating them into instructions.⁷ Because the resulting FMA tests are highly regular, we then synthesize the test program algorithmically by a software routine. The synthesized test program is highly modularized and very small. Experimental results have shown that a test program as small as 3 Kbytes can detect all crosstalk defects on the bus from the processor core to the target core.

Next, we apply the synthesized test program to the bus from the processor core, and the input buffers of the destination core capture the responses at the other end of the bus. The processor core must read back such responses to determine whether any faults on the bus occurred. However, because the processor core cannot read input buffers of a nonmemory core, we suggest a DFT scheme that lets the processor core directly observe the input buffers. The DFT circuitry consists of bypass logic added to each I/O core to improve its testability.

With the DFT support on the target I/O core, the test generation procedure first synthesizes instructions to set the target core to bypass mode, then it continues synthesizing instructions for the FMA tests. The test generation procedure does not depend on the target core's functionality.

Self-testing nonprogrammable IP cores

Testing nonprogrammable cores on SoCs is a complex problem, with many unresolved issues.⁹ Industry initiatives such as the IEEE P1500 Working Group provide some solutions, but they do not address the requirements of at speed testing.

Huang, Iyer, and Cheng have proposed a self-testing approach for a SoC's nonprogrammable cores, in which a test program running on the embedded processor delivers test patterns at speed, to other IP cores.⁹ The test patterns can be generated on the processor itself or fetched

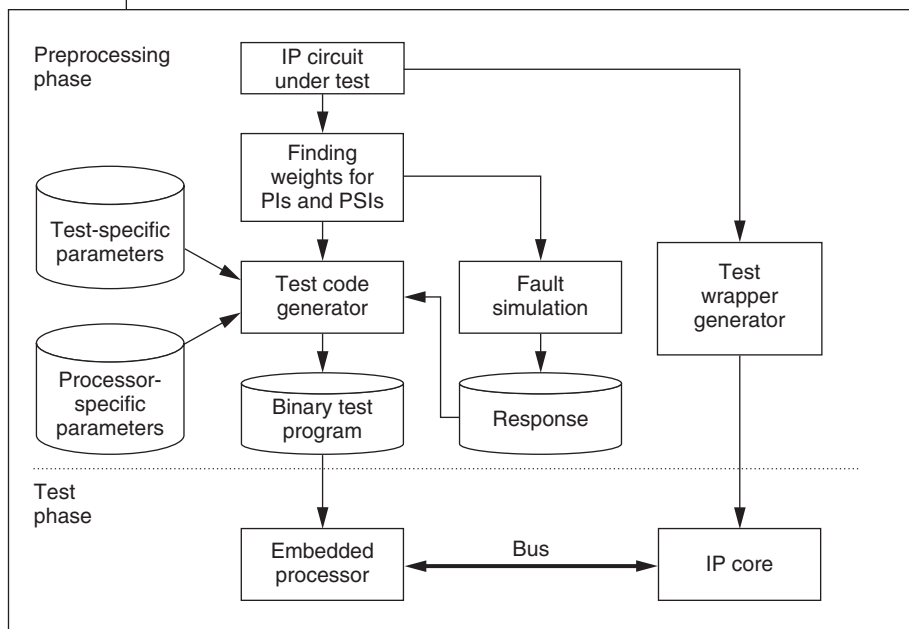


Figure 3. Self-testing nonprogrammable cores in a SoC.

from an external ATE and stored in on-chip memory. Fetching them from an ATE alleviates the need for dedicated test circuitry for pattern generation and response analysis. The approach scales to large IP cores whose structural netlists are available. Because the pattern delivery takes place at the SoC’s operational speed, this method supports delay test. A test wrapper (as in Figure 1) around each core supports pattern delivery (and contains the test support logic that controls scan chain shifting, buffers to store scan data and support at-speed test, and so on).

Figure 3 shows the test flow based on the embedded software self-testing methodology, divided into preprocessing and testing phases. This approach offers tremendous flexibility in the type of tests that we can apply to the IP cores as well as in the quality of the test pattern set, without entailing significant hardware overhead.

During preprocessing, a test wrapper configured to meet the specific testing needs is automatically inserted around the IP core under test. The IP core is then fault simulated with different sets of patterns. The method of Huang et al. for testing delay faults employs weighted random patterns generated using multiple weight sets or using multiple capture cycles after each scan sequence.⁹ Compared to pseudorandom testing, these patterns achieve the desired fault coverage

with shorter test length. Also, because they are software generated, they do not incur hardware overhead, as weighted random testing in hardware BIST does.

Next, a high-level test program is generated. This program synchronizes the software pattern generation, the start of the test application, and the analysis of the test response. The program can also synchronize testing of multiple cores in parallel. The test program is then compiled to generate processor-specific binary code.

In the test phase, the processor core runs the test program to test various IP cores. The processor core sends a test packet to the IP core test wrapper, informing it about the test application scheme—for example,

single- or multiple-capture cycle. The processor core then sends data packets to load the scan buffers and the primary input/primary output (PI/PO) buffers. The test wrapper applies the required number of scan shifts, and captures the test response for the programmed number of functional cycles. The test results are stored in the PI/PO and scan buffers, from which the processor core then reads them.

Instruction-level DFT: Test instructions

Self-testing a SoC’s manufacturing defects by using a programmable core to run test programs has several potential benefits, including at-speed testing, low DFT overhead due to the elimination of dedicated test circuitry, and better power and thermal management during testing. However, such a self-test strategy might require a lengthy test program and might not achieve adequate fault coverage. We can alleviate these problems by adding test instructions to an on-chip programmable core such as a microprocessor core. This methodology is called *instruction-level DFT*.^{10,11}

Instruction-level DFT is less intrusive than gate-level DFT techniques, which attempt to create a separate test mode somewhat orthogonal to the functional mode. If we design the test

instructions carefully so that their microinstructions reuse the datapath for the functional instructions, the overhead, which would occur only in the controller, should be relatively low. In addition, compared to the existing logic BIST approaches, this methodology is more attractive, both for applying at-speed tests and for power and thermal management during test.

The instruction-level DFT approach described by Bieker and Marwedel adds instructions to control the exceptions, such as microprocessor interrupts and resets.¹⁰ With the new instructions, the test program can achieve fault coverage between 87% and 90% for stuck-at faults. However, this approach cannot achieve higher coverage, because the test program is synthesized based on a random approach and cannot effectively control or observe some internal registers with low testability.

Our instruction-level DFT methodology systematically adds test instructions to an on-chip processor core to improve its self-testability, reduce the self-test program's size, and reduce the test application time.¹¹ To decide which instructions to add, we first analyze the processor's testability. If we identify a register in the processor as difficult to access, we add a test instruction allowing direct access of that register. We can also add test instructions to optimize the test program size and runtime.

Adding test instructions to the programmable core does not improve the testability of the SoC's other nonprogrammable cores and therefore cannot increase their fault coverage. We can, however, optimize the test programs for the nonprogrammable cores by adding new instructions. In other words, the same set of test instructions that we added to self-test the programmable cores can reduce the size and runtime of the test programs for the nonprogrammable cores.

Experimental results for instruction-level DFT on the Parwan and DLX processors show that test instructions can significantly reduce the program size and runtime by 20%, at the cost of 1.6% area overhead.

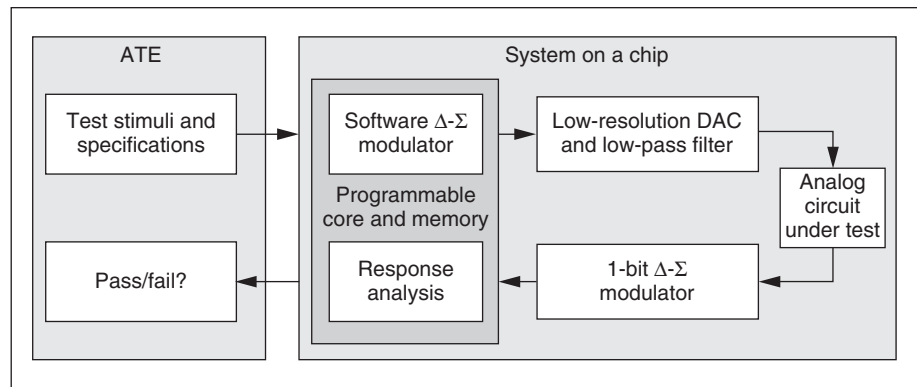


Figure 4. DSP-based self-test for analog and mixed-signal parts.

Self-testing mixed-signal and analog components, using DSP

Because most analog and mixed-signal circuits are functionally tested, analog and mixed-signal testing need expensive ATE for analog stimulus generation and response acquisition. With the advent of CMOS technology, DSP-based BIST becomes a viable solution: The signal processing required to make the pass or fail decision can take place in the digital domain with digital resources.

Huang and Cheng have proposed an efficient BIST architecture, shown in Figure 4, for testing on-chip analog and mixed-signal components.⁵ This architecture uses the Δ - Σ modulation technique for both stimulus generation and response analysis.¹²

A software Δ - Σ modulator converts the desired signal to a 1-bit digital stream. A 1-bit digital-to-analog converter (DAC) transfers the digital 1s and 0s to two discrete analog levels. Then a low-pass filter removes the out-of-band high-frequency modulation noise, thus restoring the original waveform. In practice, we extract a segment from the Δ - Σ output bitstream that contains an integer number of signal periods. The extracted pattern is stored in on-chip memory and periodically applied to the low-resolution DAC and low-pass filter to generate the desired stimulus. Similarly, for response analysis, we can insert a 1-bit Σ - Δ modulator to convert the output response of the analog device under test into a 1-bit stream, which the on-chip DSP or microprocessor cores then analyze by DSP operations.

If abundant on-chip digital programmable resources are available (as in Figure 4), on-

chip DSP or microprocessor cores can perform the software part of this technique. Otherwise, external digital equipment can provide these functions.

EMBEDDED SOFTWARE-BASED SELF-TESTING can alleviate many of the current problems with external tester-based and hardware BIST testing techniques for SoCs. One of the main tasks in applying the techniques described in this article is extracting the functional constraints in the process of test program synthesis—that is, deriving tests that can be delivered by processor instructions. Future research in this area must address the problem of automating the constraint extraction process to make the proposed solutions feasible for general processors. Using DSP-based testing techniques, Σ - Δ modulation principles, and some low-cost analog and mixed-signal DFT, the software-based self-testing paradigm can be generalized for testing analog and mixed-signal components. ■

■ References

1. L. Chen and S. Dey, "Software-Based Self-Testing Methodology for Processor Cores," *Trans. CAD*, vol. 20, no. 3, Mar. 2001, pp. 369-380.
2. W.-C. Lai, A. Krstic, and K.-T. Cheng, "On Testing the Path Delay Faults of a Microprocessor Using its Instruction Set," *Proc. VLSI Test Symp.*, IEEE CS Press, Los Alamitos, Calif., 2000, pp. 15-22.
3. W.-C. Lai, A. Krstic, and K.-T. Cheng, "Test Program Synthesis for Path Delay Faults in Microprocessor Cores," *Proc. Int'l Test Conf. (ITC 00)*, IEEE Press, Piscataway, N.J., 2000, pp. 1080-1089.
4. L. Chen and S. Dey, "Software-Based Diagnosis for Processors," *Proc. Design Automation Conf. (DAC 02)*, ACM Press, New York, 2002, pp. 259-262.
5. J.L. Huang and K.T. Cheng, "A Sigma-Delta Modulation Based BIST Scheme for Mixed-Signal Circuits," *Proc. Asia and South Pacific Design Automation Conf.*, IEEE Press, Piscataway, N.J., 2000, pp. 605-610.
6. L. Chen, X. Bai, and S. Dey, "Testing for Interconnect Crosstalk Defects Using On-Chip Embedded Processor Cores," *Proc. Design Automation Conf. (DAC 01)*, ACM Press, New York, 2001, pp. 317-320.
7. W.-C. Lai, J.-R. Huang, and K.-T. Cheng, "Embedded-Software-Based Approach to Testing Crosstalk-Induced Faults at On-Chip Buses," *Proc. VLSI Test Symp. (VTS 01)*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 204-209.
8. M. Cuviallo et al., "Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects," *Proc. Int'l Conf. Computer-Aided Design (ICCAD 99)*, ACM Press, New York, 1999, pp. 297-303.
9. J.-R. Huang, M.K. Iyer, and K.-T. Cheng, "A Self-Test Methodology for IP Cores in Bus-Based Programmable SoCs," *Proc. VLSI Test Symp. (VTS 01)*, IEEE CS Press, Los Alamitos, Calif., 2001, pp. 198-203.
10. U. Bieker and P. Marwedel, "Retargetable Self-Test Program Generation Using Constraint Logic Programming," *Proc. Design Automation Conf. (DAC 95)*, ACM Press, New York, 1995, pp. 605-611.
11. W.-C. Lai and K.-T. Cheng, "Instruction-Level DFT for Testing Processor and IP Cores in System-on-a-Chip," *Proc. Design Automation Conf. (DAC 01)*, ACM Press, New York, 2001, pp. 59-64.
12. B. Dufort and G.W. Roberts, "Signal Generation Using Periodic Single- and Multi-Bit Sigma-Delta Modulated Streams," *Proc. Int'l Test Conf. (ITC 97)*, IEEE Press, Piscataway, N.J., 1997, pp. 396-405.



Angela Krstic is a research scientist at the University of California, Santa Barbara. Her research interests include delay testing, deep-submicron testing, and SoC testing. Krstic has a BS in electrical engineering from the University of Belgrade, Yugoslavia, and an MS and PhD in electrical and computer engineering from the University of California, Santa Barbara.



Li Chen is a PhD candidate at the University of California, San Diego. Her research interests include self-test and self-diagnosis of microprocessor cores. Chen has a BS and MS in electrical and computer engineering from Carnegie Mellon University.



Wei-Cheng Lai works at the University of California, San Diego. His research interests include system-on-a-chip testing and delay testing. Lai has a BSEE from

National Taiwan University, an MSEE from the University of Southern California, and a PhD in electrical and computer engineering from the University of California, Santa Barbara.



Kwang-Ting Cheng is a professor and director of the Computer Engineering Program at the University of California, Santa Barbara, and is associate EIC of *IEEE Design & Test*. His current research interests include VLSI testing, design verification, and multimedia computing. Cheng has a BS in electrical engineering from National Taiwan University and a PhD in electrical engineering and computer science from the University of California, Berkeley. He is a Fellow of the IEEE.



Sujit Dey is a professor in the Electrical and Computer Engineering Department at the University of California, San Diego. His research interests include the devel-

opment of configurable platforms involving adaptive wireless protocols and algorithms, and deep-submicron adaptive SoCs for next-generation wireless appliances and network infrastructure devices. Dey has a PhD in computer science from Duke University.

■ Direct questions and comments about this article to Angela Krstic, Dept. of Electrical and Computer Engineering, Univ. of California, Santa Barbara, CA 93106; angela@windcave.ece.ucsb.edu.

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

Get access

to individual IEEE Computer Society documents online.

More than 57,000 articles
and conference papers available!

US\$5 per article for members

US\$10 for nonmembers

<http://computer.org/publications/dlib/>

