Embedded Software-Based Self-Testing for SoC Design

A. Krstic[1] W.-C. Lai[1] L. Chen[2] K.-T. Cheng[1] S. Dey[2]

[1] University of California, Santa Barbara, CA 93106, {angela, wlai, timcheng}@windcave.ece.ucsb.edu

[2] University of California, San Diego, CA 92037, {lichen, dey}@ece.ucsd.edu

ABSTRACT

At-speed testing of high-speed circuits is becoming increasingly difficult with external testers due to the growing gap between design and tester performance, growing cost of high-performance testers and increasing yield loss caused by inherent tester inaccuracy. Therefore, empowering the chip to test itself seems like a natural solution. Hardware-based self-testing techniques have limitations due to performance and area overhead and problems caused by the application of non-functional patterns.

Embedded software-based self-testing has recently become focus of intense research. In this methodology, the programmable cores are used for on-chip test generation, measurement, response analysis and even diagnosis. After the programmable core on a System-on–Chip (SoC) has been self-tested, it can be reused for testing on-chip buses, interfaces and other non-programmable cores. The advantages of this methodology include at-speed testing, low design-for-testability overhead and application of functional patterns in the functional environment. In this paper, we give a survey and outline the roadmap and challenges of this emerging embedded software-based self-testing paradigm.

Categories and Subject Descriptors

B.8.1 [**Integrated Circuits**]: Performance and Reliability – *reliability, testing and fault-tolerance.*

General Terms

Algorithms, Performance, Reliability.

Keywords

VLSI test, SoC test, functional test, microprocessor test.

1. INTRODUCTION

System-on-chip (SoC) has become a widely accepted architecture for highly complex systems on a single chip. An SoC contains a large number of complex, heterogeneous components that can include digital, analog, mixed-signal, radio frequency (RF), micromehanical and other systems on a single piece of silicon. The increasing heterogeneity and programmability associated with the system-on-chip architecture together with the rapidly increasing operating frequencies and technology changes are demanding fundamental changes in VLSI testing.

This work was supported in part by the MARCO/DARPA Gigascale Silicon Research Center (http://www.gigascale.org). Their support is gratefully acknowledged.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC 2002, June 10-14, 2002, New Orleans, Louisiana, USA.
Copyright 2002 ACM 1-58113-461-4/02/0006…$5.00.

The test application using testers poses challenges because the testers performance is increasing at a slower rate than the device speed. This translates into an increasing yield loss due to external testing since guardbanding to cover tester errors results in a loss of more and more good chips. In addition, high-speed testers are very costly. Also, for mixed-signal testing, analog instrumentation needed for precise analog testing is not present on the digital logic tester. Having to test mixed-signal chips would require a mixed-signal tester that is even more expensive and implies a two-pass testing strategy. However, there are also cases (such as those where vast amount of digital data has to be pumped through an analog interface e.g., 3GIO, Infiniband, SATA to test the digital core) where neither of the two platforms would work.

Built-in self-test (BIST) solutions eliminate the need for high speed testers and offer the ability to apply and analyze at-speed test signals on chip with greater accuracy than that of the tester. Existing BIST techniques belong to the class of structural BIST. Structural BIST, such as scan-based BIST techniques [1][2][3], offer good test quality but require addition of dedicated test circuitry. Therefore, they incur non-trivial area, performance and design time overhead. Moreover, structural BIST applies non-functional, high-switching random patterns and thus, causes much higher power consumption than normal system operations. Also, to apply at-speed tests to detect timing related faults, existing structural BIST needs to resolve various complex timing issues related to multiple clock domains, multiple frequencies and test clock skews that are unique in the test mode.

A new *embedded software-based self-testing* paradigm [4][5][6] has a potential to alleviate the problems due to the use of external testers as well as embedded hardware tester problems described above. In this testing strategy, it is assumed that programmable cores on the SoC (such as processor and DSP cores) are first self-tested by running an automatically synthesized test program which can achieve high fault coverage. This allows application of functional tests in the normal functional operating environment of the design. Thus, it eliminates problems caused by application of non-functional patterns as well as problems caused by non-functional environment during test application. Next, the programmable core can be used as a pattern generator and response analyzer to test on-chip buses, interfaces between cores or even other cores including digital, mixed-signal and analog components on an SoC. This solution is sometimes also referred to as *functional self-testing*.

The concept of embedded software-based self-testing is illustrated in Figure 1 using a bus-based SoC. The IP cores in the SoC are connected to a Peripheral Interconnect (PCI) [7] bus via the virtual component interface (VCI) [8]. The VCI acts as a standard communication interface between the IP core and the on-chip bus. First, the microprocessor tests itself by executing a set of instructions. Next, the processor can be used for testing the bus and other non-programmable IP cores in the SoC. In order to support the self-testing methodology, the IP core has a test wrapper around it. The test wrapper contains test support logic needed to control shifting of the scan chain, buffers to store scan data and support at-speed test, etc. In this example, the on-chip bus is a shared bus and the arbiter controls access to the bus.

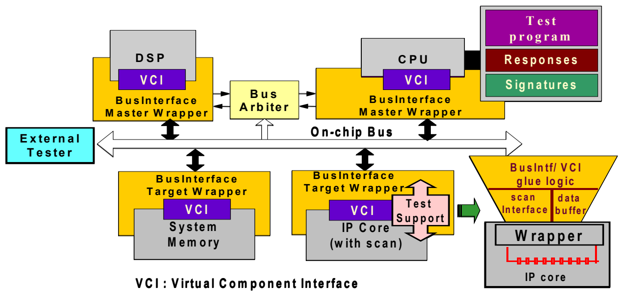

Figure 1: Embedded software-based self-testing for SoC.

There are several advantages of the embedded software-based self-test approach. First, it allows reuse of programmable resources on SoCs for test purposes. In other words, this strategy views testing as an application of the programmable components in the SoC and thus, minimizes the addition of dedicated test circuitry for design-for-testability or self-test.

Second, in addition to eliminating the need for costly high-speed testers, it can also reduce the yield loss due to tester accuracy problems. Self-testing offers the ability to apply and analyze at-speed test signals on chip with accuracy greater than that available with the tester.

Third, while the hardware-based self-test must be applied in the non-functional BIST mode, software-based self-test can be applied in the normal operational mode of the design, i.e., the tests are applied by executing instruction sequences as in regular system operations. This eliminates the problems created by application of non-functional patterns that can result in excessive power consumption when hardware BIST is used.

Also, functional self-test can alleviate some delay testing problems. Delay testing using scan techniques requires turning all state elements into scannable counterparts and they can be loaded serially through the scan port. The scan shifting is usually done at a lower frequency to make for easier Design-for-Testability (DfT) implementation. After the proper states are all scanned-in, the system clock is initiated for 2 or more cycles. The first clock is the *launch clock* that launches the output of the state elements into the combinational blocks. The subsequent clock (*capture clock*) captures the responses back into the scan chain for verification. During the brief launch/capture system clock, large current surges will happen affecting the circuit delays. Increasing the clock period would allow for defect based testing, but this would not guarantee that the circuit would meet the rated speed. Another possible solution would be to launch multiple system clocks after the scan shift so that the power grid has a chance to stabilize. This requires hundreds of cycles and presents a big problem for ATPG.

Functional testing can also relieve problems due to application of non-functional patterns during structural delay testing through AC-scan or BIST that can result in over-testing and yield loss. Experiments have shown that many structurally testable delay faults in the microprocessors can never be sensitized in the functional mode of the circuit [5]. This is because no functionally applicable vector sequence can excite these delay faults and propagate the fault effects into destination outputs/flip-flops at-speed. Defects on these faults will not affect the circuit performance and their testing is not necessary.

Testing of analog circuits has been a costly process because of the limited access to the analog parts and testers needed for functional testing. The situation has become worse due to integrating various digital and analog cores onto the SoC, in which testing the analog parts becomes the bottleneck of production testing. Self-testing of on-chip ADC/DAC and analog components using DSP-based approaches can alleviate these problems.

In this paper, we give a survey of the embedded software-based self-testing methods. We start by discussing processor self-test methods targeting stuck-at faults and delay faults. Next, we continue with self-testing of buses and global interconnects and well as other non-programmable IP cores on SoC. We also describe instruction-level DfT methods based on insertion of test instructions to increase the fault coverage and reduce the test application time and test program size. Finally, we summarize DSP-based self-test for analog/mixed-signal components.

2. EMBEDDED PROCESSOR SELF-TEST

While logic BIST may perform well on industrial application specific integrated circuits (ASICs), it is less feasible on microprocessors. First, the design changes needed for making a microprocessor BIST-ready (e.g., immune to problems such as bus contentions even when pseudorandom test patterns are applied) may come with unacceptable cost, such as substantial manual effort and significant performance degradation. In addition, microprocessors are especially random pattern-resistant. Due to timing-critical nature of microprocessors, test points may not be acceptable as a solution to this problem, as they could introduce performance degradation on critical paths. Deterministic BIST, on the other hand, may lead to unacceptable area overhead, as the size of the on-chip hardware for encoding deterministic test patterns depends on the circuit testability [9].

A number of approaches have been proposed to generate functional tests for microprocessors [10][11][12][13][14]. Some propose to apply the tests with external testers [10], while others allow the processors to tests themselves with self-test programs [11][12][13][14]. A common characteristic of approaches in [11][12] is application of randomized instructions to the processor under test. 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. Approaches in [13][14] use structural ATPG to generate tests for stuck-at faults in the processor. They use the RTL information of the processor to form a set of RTL-module equations that can realize the generated gate-level test. Solving the set of equations specifies the required instruction sequences and operands. All of the above approaches target only stuck-at faults and the methods cannot be easily generalized for delay faults.

Unlike hardware-based self-testing, software-based testing is non-intrusive since it applies tests in the normal operational mode of the circuit. Moreover, software instructions have the ability of guiding the test patterns through a complex processor, avoiding the blockage of the test data due to non-functional control signals as in the case of hardware-based logic BIST.

Embedded software-based self-test methods for processors have been proposed in [4][5][6]. These methods consist of two steps: the test preparation step and the self-testing step. The test preparation step involves generation of *realizable* tests for components of the processor. Realizable tests are those that can be delivered using instructions. Therefore, to avoid producing undeliverable test patterns, the tests are generated under the constraints imposed by the processor instruction set. The tests can then be either stored or generated on-chip, depending on which method is more efficient for a particular case. A low-speed tester can be used to load the self-test signatures or the predetermined tests to the processor memory prior to the application of tests.

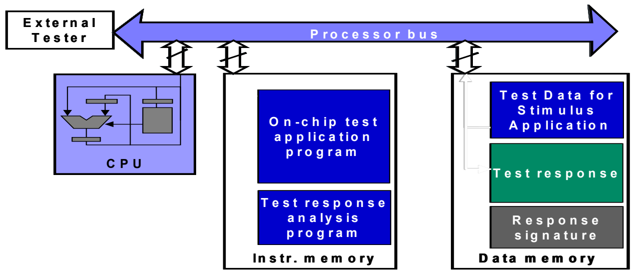
External Tester
Processor bus
CPU
On-chip test application program
Test response analysis program
Instr. memory
Test Data for Stimulus Application
Test response
Response signature
Data memory

Figure 2: Embedded processor self-testing.

The self-testing step, illustrated in Figure 2, involves the application of these tests using a software tester. The software tester can also compress the responses into self-test signatures that can then be stored into the memory. The signatures can later be unloaded and analyzed by an external tester. The assumption here is that the processor memory has been tested with standard techniques such as memory BIST before the application of the test and the memory is assumed to be fault-free.

In the following, we describe the embedded software-based self-test method for testing stuck-at [4] and path delay faults [5][6] in microprocessors using their instruction set.

Stuck-at Fault Testing. As the first step, the realizable structural tests for components of the processor are developed. Component tests can either be stored or generated on-chip. If tests are generated on-chip, the test needs of each component are characterized by a self-test signature, which includes the seed S and the configuration C of a pseudorandom number generator as well as the number of test patterns to be generated N. The self-test signatures can be expanded on-chip into test sets using a pseudorandom number generation program. Multiple self-test signatures may be used for one component if necessary. Thus, this self-test methodology allows incorporation of any deterministic BIST techniques that encode a deterministic test set as several pseudorandom test sets [15][16].

By targeting the structural test needs of less complex components, the proposed method has the fault coverage advantage of deterministic structural testing. Since component test application and response collection are done with instructions instead of with scan chains, it requires no area or performance overhead and the test application is performed at-speed. Most importantly, by shifting the role of external testers to applying test programs and unloading responses, it enables at-speed testing of gigahertz processors with low-speed testers.

Using manually extracted constraints, the above scheme has been applied to a simple Parwan processor [17]. The results have demonstrated the feasibility and effectiveness of the software-based self-test method by generating a high-coverage test program for the simple processor.

Delay Testing. To synthesize a test program for self-test of path delay faults in a microprocessor using its instructions, first the spatial and temporal constraints between and at the registers and control signals are extracted (given the instruction set architecture and the micro-architecture of the processor core). Next, a path classification algorithm, extended from [18][19], 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 universe. This helps reduce the computational effort of the subsequent test generation process. As the experimental results in [5] show, a high percentage of the paths in Parwan processor [17] and DLX processor [20] are functionally untestable.

Next, a subset of long paths among the functionally testable paths are selected as targets for test generation. A gate-level ATPG for path delay faults is extended to incorporate the extracted constraints into the test generation process and it is used to generate test vectors for each target path delay fault. If the test is successfully generated, it not only sensitizes the path but it also meets the extracted constraints. Therefore, it is most likely to be deliverable by instructions (if the complete set of constraints has been extracted, the delivery by instructions could be guaranteed). 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, a pre-defined propagating routine is used to propagate the fault effects captured in the registers/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. The procedure is repeated until all target faults have been processed. The test program is generated off-line and later used to test the microprocessor at-speed.

To apply the synthesized test program, it is loaded into the on-chip memory by an external low speed tester. When the test program is being executed at-speed, a set of signatures is recorded in memory. At the end of the test program, a response analysis subroutine is called to further compress the recorded signatures in memory and finally, compare the compressed signature with the correct signature.

This test synthesis program has been applied to Parwan [17] and DLX [20] processors. On the average, 5.3 and 5.9 instructions were needed to deliver a test vector and the achieved fault coverage for *testable* path delay faults was 99.8% and 96.3% for Parwan and DLX, respectively.

3. SELF-TESTING OF BUSES AND GLOBAL INTERCONNECTS

In SoC designs a large amount of core-to-core communications must be realized with long interconnects. As gate delay continues to decrease, the performance of interconnect is becoming increasingly important in achieving a high overall performance. However, due to the increase of cross-coupling capacitance and mutual inductance, signals on neighboring wires may interfere with each other, causing excessive delay or loss of signal integrity. While many techniques have been proposed to reduce crosstalk, due to the limited design margin and unpredictable process variations, the crosstalk must also be addressed in manufacturing testing.

Due to its timing nature, testing for crosstalk effects needs to be conducted at the operational speed of the circuit-under-test. However, at-speed testing of GHz systems requires prohibitively costly high-speed testers. Moreover, with external testing, hardware access mechanisms are required for applying tests to interconnects deeply embedded in the system. This may lead to unacceptable area or performance overhead.

A BIST technique in which an SoC tests its own interconnects for crosstalk defects using on-chip hardware pattern generators and error detectors has been proposed in [21]. Although the amount of area overhead may be amortized for large systems, for small systems, the amount of relative area overhead may beb unacceptable. Moreover, hardware-based self-test approaches, as the one in [21], may cause over-testing and yield loss, as not all test patterns generated in the test mode are valid in the normal operational mode of the system.

The problem of testing system-level interconnects in embedded processor-based SoCs, which are the most dominant type of SoCs, has been addressed in [22][23]. In such SoCs, most of the system-level interconnects, such as the on-chip buses, are accessible to the embedded processor core(s). The proposed methodology is software-based and enables an embedded processor core in SoC to test for crosstalk effects in these interconnects by executing a software program. The strategy is to let the processor execute a self-test program with which the test vector pairs can be applied to the appropriate bus in the normal functional mode of the system. In the presence of crosstalk-induced glitch or delay effects, the second vector in the vector pair becomes distorted at the receiver end of the bus. The processor, can then store this error effect to the memory as a test response, which can be later unloaded by an external tester for off-chip analysis.

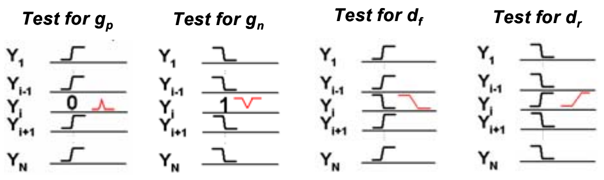

Figure 3: Maximal aggressor tests for victim Y_i.

Maximum Aggressor (MA) fault model proposed in [24] is suitable for modeling crosstalk defects on interconnects. It 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 other N-1 wires act as aggressors. The required transitions on the aggressor/victim lines to excite the four error types are shown in Figure 3. For a set of N interconnects, there are $4N$ MA faults, requiring $4N$ MA tests. It has been shown in [24] that these $4N$ faults cover all crosstalk defects on any of the N interconnects.

Chen *et al.* [22] concentrate on testing data and address bus in a processor-based SoC. The crosstalk effects on the interconnects are modeled using the MA fault model.

Testing Data Bus. For a bi-directional bus such as data bus, crosstalk effects vary as the bus is driven from different directions. Thus crosstalk tests need to be conducted in both directions [21].

To apply a test vector pair (*v1*, *v2*) for the data bus from an SoC core to the CPU, the CPU first exchanges data *v1* with the core. The direction of data exchange is irrelevant. For example, if the core is the memory, the CPU may either read *v1* from the memory or write *v1* to the memory. The CPU then requests data *v2* from the core (a memory-read if the core is memory). Upon the arrival of *v2*, the CPU writes *v2* to memory for later analysis.

To apply a test vector pair (*v1*, *v2*) to the data bus from the CPU to an SoC core, the CPU first exchanges data *v1* with the core. Then, the CPU sends data *v2* to the core (a memory-write if the core is memory). If the core is memory, *v2* can be directly stored to an appropriate address for later analysis. Otherwise, the CPU must execute additional instructions to retrieve *v2* from the core and store it to memory.

Testing Address Bus. To apply a test vector pair (*v1, v2*) to the address bus, which is a unidirectional bus from the CPU to an SoC core, the CPU first requests data from two addresses (*v1* and *v2*) in consecutive cycles. In the case of a non-memory core, since the CPU addresses the core via memory-mapped I/O, *v2* must be the address corresponding to the core. If *v2* is distorted by crosstalk, the CPU would be receiving data from a wrong address, *v2'*, which may be a physical memory address or an address corresponding to a different core. By keeping different data at *v2* and *v2'* (i.e., mem[*v2*] $\neq$ mem[*v2'*]), the CPU is able to observe the error and store it to memory for analysis.

The feasibility of this method has been demonstrated by applying it to test the interconnects of a processor-memory system. The defect coverage was evaluated using a system-level crosstalk defect simulation method.

Functionally Maximal Aggressor Tests. Even though the MA tests have been proven to cover all physical defects related to crosstalk between interconnects, Lai *et al.* [23] observe that many of them can never occur during normal system operation due to constraints imposed by the system. Therefore, testing buses using MA tests might screen out chips that are functionally correct under any pattern produced under normal system operation. Instead, *Functionally Maximal Aggressor (FMA) tests* meeting the system constraints and being possible to be delivered under the functional mode are proposed [23]. The tests can be synthesized 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 3K bytes can detect all crosstalk defects on the bus from the processor core to the target core.

The synthesized test program is applied 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. Such responses need to be read back by the processor core to determine whether or not any faults on the bus occurred. However, because the input buffers of a non-memory core cannot be read by the processor core, a DfT scheme is suggested to allow direct observability of the input buffers by the processor core. 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 the bypass mode and then it continues with synthesizing instructions for the FMA tests. The test generation procedure does not depend on the functionality of the target core.

4. SELF-TESTING OF OTHER NON-PROGRAMMABLE IP-CORES

Testing non-programmable cores on an SoC is a complex problem with many unresolved issues [25]. Industry initiatives such as the IEEE P1500 Working Group [26] provide some solutions for IP core testing. However, they do not address the requirements of at-speed testing.

A self-testing approach for non-programmable cores on an SoC has been proposed in [25]. In this approach, a test program running on the embedded processor delivers test patterns to other IP cores in the SoC at-speed. The test patterns can be generated on the processor itself or fetched from an external ATE and stored in on-chip memory. This alleviates the need for dedicated test circuitry for pattern generation and response analysis. The approach is scalable to large size IP cores whose structural netlists are available. Since the pattern delivery is done at the SoC operational speed, it supports delay test. A test wrapper (shown in Figure 1) is inserted around each core to support pattern delivery. It contains test support logic needed to control shifting of the scan chain, buffers to store scan data and support at-speed test, etc.

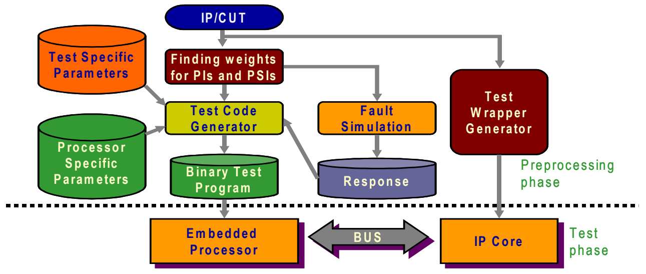

Figure 4: The test flow.

The test flow based on the embedded software self-testing methodology is illustrated in Figure 4. It offers tremendous flexibility in the type of tests that can be applied to the IP cores well as in the quality of the test pattern set without entailing significant hardware overhead. Again, the flow is divided into a pre-processing phase and a testing phase.

In the pre-processing phase, a test wrapper is automatically inserted around the IP core under test. The test wrapper is configured to meet the specific testing needs for the IP core. The IP core is then fault simulated with different sets of patterns. Weighted random patterns generated with multiple weight sets or using multiple capture cycles [3] after each scan sequence are used in [25]. Next, a high-level test program is generated. This program synchronizes the software pattern generation, start of the application of the test and analysis of the test response. The program can also synchronize testing multiple cores in parallel. The test program is then compiled to generate a processor specific binary code.

In the test phase, the test program is run on the processor core to test various IP cores. A test packet is sent to the IP core test wrapper informing it about the test application scheme (e.g., single or multiple capture cycle). Data packets are then sent to load the scan buffers and the 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 results of the test are stored in the PI/PO buffers and the scan buffers and from there they are read out by the processor core.

5. INSTRUCTION LEVEL DFT

While self-testing manufacturing defects in an SoC by running test programs using a programmable core has many potential benefits, such a self-test strategy might require a lengthy test program and might not achieve a high enough fault coverage. These problems can be alleviated by applying a DfT methodology based on adding test instructions to an on-chip programmable core such as a microprocessor core. This methodology is called *instruction-level DfT*.

Instruction-level DfT inserts test circuitry in the form of test instructions and should be a less intrusive approach as compared to the gate-level DfT techniques which attempt to create a separate test mode somewhat orthogonal to the functional mode. If the test instructions are carefully designed such that their micro-instructions reuse the datapath for the functional instructions and do not require any new datapath, the overhead, which only occurs in the controller, should be relatively low.

Instruction-level DfT methods have been proposed in [11] [27]. The approach in [11] adds instructions to control the exceptions, e.g., microprocessor interrupts and reset. With the new instructions, the test program can achieve fault coverage between 87% and 90% for stuck-at faults. However, this approach cannot achieve a higher coverage because the test program is synthesized based on a random approach and it is not able to effectively control or observe some internal registers that have low testability.

The DfT methodology proposed in [27] systematically adds test instructions to an on-chip processor core to improve the self-testability of a processor core, reduce the size of the self-test program as well as reduce its run time (i.e., reduce the test application time). To decide which instructions to add, the testability of the processor is analyzed first. If a register in the processor is identified as hard-to-access, a test instruction allowing direct accessing of the register is added. In addition to these test instructions, test instruction can be also added to optimize the test program size and run time.

Adding test instructions to the programmable core does not improve the testability of other non-programmable cores on the SoC. Therefore, instruction-level DfT cannot increase the fault coverage of the non-programmable cores. However, the test programs for testing the non-programmable cores can be optimized by adding new instructions. In other words, the same set of test instructions added for self-testing the programmable cores can be used to reduce the size and run time of the test programs for testing other non-programmable cores.

The experimental results on two processors (Parwan [17] and DLX [20]) show that test instructions can reduce the program size and program run time by 20% at the cost of 1.6% area overhead.

6. ON-CHIP ADC/DAC AND ANALOG COMPONENTS

For mixed-signal systems integrating both analog and digital functional blocks onto the same chip, testing of analog/mixed-signal parts has become the bottleneck during production testing. Because most analog/mixed-signal circuits are functionally tested, analog/mixed-signal testing needs expensive automatic test equipment (ATE) for analog stimulus generation and response acquisition. One promising solution to this problem is BIST that utilizes on-chip resources (either shared with functional blocks or dedicated BIST circuitry) to perform on-chip stimulus generation and response acquisition. Under the BIST approach, the requirement on the external test equipment is less stringent. Furthermore, stimulus generation and response acquisition is more immune to environmental noise during the test process.

With the advent of the CMOS technology, DSP-based BIST becomes a viable solution for analog/mixed-signal systems as the required signal processing to make the pass/fail decision can be realized in the digital domain with digital resources.

An efficient BIST architecture for testing on-chip analog and mixed-signal components has been proposed in [28]. It employs the delta-sigma modulation technique for both stimulus generation

[29] and response analysis. Figure 5 illustrates this delta-sigma modulation-based BIST architecture.

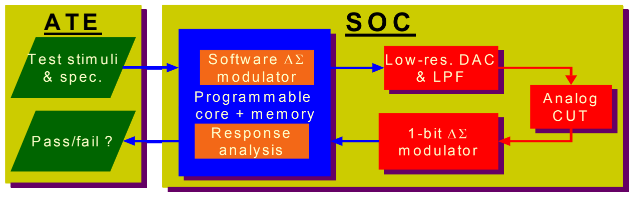

Figure 5: DSP-based self-test for analog/mixed-signal parts.

A *software* delta-sigma modulator converts the desired signal to one-bit digital steam. The digital 1's and 0's are then transferred to two discrete analog levels by one-bit DAC followed by a low-pass filter that removes the out-of-band high-frequency modulation noise, and thus restores the original waveform. In practice, one extracts a segment from the delta-sigma output bit stream 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, a 1-bit $\Sigma - \Delta$ modulator can be inserted to convert the analog DUT output response into a 1-bit stream which is then analyzed by digital signal processing (DSP) operations performed by on-chip DSP/microprocessor cores. Among the 1-bit $\Sigma - \Delta$ modulation architectures, the 1st-order configuration is the most stable and has the maximal input dynamic range. However, it is not practical for high-resolution applications (as rather high over sampling rate will be needed), and suffers inter-modulation distortion. Compared to the 1st-order configuration, the 2nd-order configuration has a smaller dynamic range but is more applicable for high-resolution applications.

Note that the software part of this technique, i.e., the software $\Sigma - \Delta$ modulator and the response analyzer, can be performed by on-chip DSP/microprocessor cores, if abundant on-chip digital programmable resources are available (as indicated in Figure 5), or by external *digital* test equipment.

7. CONCLUSIONS

Embedded software-based self-testing has a potential to alleviate many of the current external tester-based and hardware BIST testing techniques for SoCs. In this paper, we give a summary of the recently proposed techniques for self-testing for system-on-chips. One of the main tasks in applying these techniques is extracting the functional constraints in the process of test program synthesis, i.e., deriving tests that can be delivered by processor instructions. Future research in this area must address the problem of automating the constraint extraction process in order to make the proposed solutions feasible for general processors. The software-based self-testing paradigm can be further generalized for analog/mixed-signal components through the integration of DSP-based testing techniques, $\Sigma - \Delta$ modulation principles and some low-cost analog/mixed-signal DfT.

ACKNOWLEDGMENTS

We would like to thank Dr. T. M. Mak, Intel on many stimulating discussions and useful insights on the topics in this paper.

References

[1] C.-J. Lin, Y. Zorian, and S. Bhawmik. Integration of Partial Scan and Built-in Self-Test. *JETTA*, Aug. 1995.
[2] K.-T. Cheng and C.-J. Lin. Timing-Driven Test Point Insertion for Full-Scan and Partial-Scan BIST. *ITC*, 1995.
[3] H.-C. Tsai, S. Bhawmik, and K.-T. Cheng. An Almost Full-Scan BIST Solution – Higher Fault Coverage and Shorter Test Application Time. *ITC*, 1998.
[4] L. Chen and S. Dey. Software-Based Self-Testing Methodology for Processor Cores. *TCAD*, Mar. 2001.
[5] W.-C. Lai, A. Krstic, and K.-T. Cheng. On Testing The Path Delay Faults of a Microprocessor Using its Instruction Set. *VTS*, 2000.
[6] W.-C. Lai, A. Krstic, and K.-T. Cheng. Test Program Synthesis for Path Delay Faults in Microprocessor Cores. *ITC*, 2000.
[7] PCI Special Interest Group, Hillsboro, Oregon, USA. *PCI Local Bus Specification, Revision 2.0*, Apr. 1994.
[8] On-Chip Bus Development Working Group. *Virtual Component Interface Standard (OCB 2 1.0),* Mar. 2000.
[9] G. Kiefer, H. Vranken, E. J. Marinissen, and H.-J. Wunderlich. Application of Deterministic Logic BIST on Industrial Circuits. *ITC,* 2000.
[10] R. Rajsuman. Testing A System-on-Chip with Embedded Microprocessors. *ITC,* 1999.
[11] J. Shen and J. A. Abraham. Native Mode Functional Test Generation for Processors with Applications to Self Test and Design Validation. *ITC,* 1998.
[12] K. Batcher and C. Papachristou. Instruction Randomization Self Test for Processor Cores. *VTS,* 1999.
[13] J. Lee and J. H. Patel. Architectural Level Test Generation for Microporcessors. *TCAD*, 1994.
[14] J. Lee and J. H. Patel. Hierarchical Test Generation Under Architectural Level Functional Constraints. *TCAD,* Sep. 1996.
[15] S. Hellebrand and H.-J. Wunderlich. Mixed-Mode BIST Using Embedded Processors. *ITC*, 1996.
[16] R. Dorsch and H.-J. Wunderlich. Accumulator Based Deterministic BIST. *ITC,* 1998.
[17] Z. Navabi. *VHDL: Analysis and Modeling of Digital Systems.* McGraw-Hill, New York, 1997.
[18] K.-T. Cheng and H.-C. Chen. Classification and Identification of Nonrobustly Untestable Path Delay Faults. *TCAD*, Aug. 1996.
[19] A. Krstic, S. T. Chakradhar, and K.-T. Cheng. Testable Path Delay Fault Cover for Sequential Circuits. *EDAC*, 1996.
[20] M. Gumm. VHDL – Modeling and Synthesis of the DLXS RISC Processor. *VLSI Design Course Notes*, Univ. of Stuttgart, Germany, Dec. 1995.
[21] X. Bai, S. Dey, and J. Rajski. Self-Test Methodology for At-Speed Test of Crosstalk in Chip Interconnects. *DAC*, 2000.
[22] L. Chen, X. Bai, and S. Dey. Testing for Interconnect Crosstalk Defects Using On-Chip Embedded Processor Cores. *DAC*, 2001.
[23] W.-C. Lai, J.-R. Huang, and K.-T. Cheng. Embedded-Software-Based Approach to Testing Crosstalk-Induced Faults at On-Chip Buses. *VTS*, 2001.
[24] M. Cuviello, S. Dey, X. Bai, and Y. Zhao. Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects. *ICCAD*, 1999.
[25] J.-R. Huang, M. K. Iyer and K.-T. Cheng. A Self-Test Methodology for IP Cores in Bus-Based Programmable SoCs. *VTS*, 2001.
[26] IEEE P1500 Web Site, http://grouper.ieee.org/groups/1500/.
[27] W.-C. Lai and K.-T. Cheng. Instruction-Level DFT for Testing Processor and IP Cores in System-on-a-Chip. *DAC*, 2001.
[28] J.L. Huang and K.T. Cheng. A Sigma-Delta Modulation Based BIST Scheme for Mixed-Signal Circuits. *ASPDAC*, 2000.
[29] B. Dufort and G. W. Roberts. Signal Generation Using Periodic Single and Multi-Bit Sigma-Delta Modulated Streams. *ITC*, 1997.