JOURNAL OF ELECTRONIC TESTING: Theory and Applications 19, 113–123, 2003

LI-BIST: A Low-Cost Self-Test Scheme for SoC Logic Cores and Interconnects*

KRISHNA SEKAR AND SUJIT DEY
Department of Electrical and Computer Engineering, University of California, San Diego, CA, USA
ksekar@ece.ucsd.edu
dey@ece.ucsd.edu

Received May 23, 2002; Revised November 7, 2002

Editor: A. Ivanov

Abstract. For system-on-chips (SoC) using deep submicron (DSM) technologies, interconnects are becoming critical determinants for performance, reliability and power. Buses and long interconnects being susceptible to crosstalk noise, may lead to functional and timing failures. Existing at-speed interconnect crosstalk test methods propose inserting dedicated interconnect self-test structures in the SoC to generate vectors which have high crosstalk defect coverage. However, these methods may have a prohibitively high area overhead. To reduce this overhead, existing logic BIST structures like LFSRs could be reused to deliver interconnect tests. But, as shown by our experiments, use of LFSR tests achieve poor crosstalk defect coverage. Additionally, it has been shown that the power consumed during testing can potentially become a significant concern.

In this paper, we present Logic-Interconnect BIST (LI-BIST), a comprehensive self-test solution for both the logic of the cores and the SoC interconnects. LI-BIST reuses existing logic BIST structures but generates high-quality tests for interconnect crosstalk defects, while minimizing the area overhead and interconnect power consumption. The application of the LI-BIST methodology on example SoCs indicates that LI-BIST is a viable, low-cost, yet comprehensive solution for testing SoCs.

Keywords: LI-BIST, BIST, crosstalk test, SoC test, low-power test

1. Introduction

Advances in device technology have led to an era where entire systems can be implemented on a single chip, referred to as System-on-Chip (SoC). As SoC complexity grows with increasing integration and reducing feature sizes, the on-chip interconnect architecture, which is responsible for inter-core communication, plays a much more critical role since it starts dominating system performance [14] and power consumption [10]. Reliability of SoCs depends increasingly on the error-free operation of such interconnects. Testing of SoCs, hence, implies testing not only the logic cores but also the interconnect architecture.

The use of deep sub-micron (DSM) technology in SoCs increases the capacitive coupling between adjacent wires leading to severe crosstalk noise, which causes the functionality or performance of the chip to deviate significantly from expected behavior. Several physical design [4, 18] and analysis [9, 12, 13] techniques have been developed to allow design for margin and to minimize signal integrity problems. However, these may be prohibitive in terms of design cost. Furthermore, it impossible to take into account all the possible process variations and physical defects

*This work was supported by MARCO/DARPA Gigascale Silicon Research Center (GSRC).

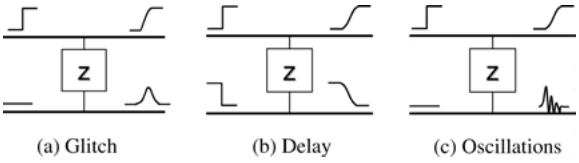

Fig. 1. Types of crosstalk faults.

during design. Hence, we need to address the crosstalk issue by means of testing techniques.

Increased cross-coupling capacitance between a pair of interconnects can produce either glitches or delays depending upon the signal transitions at the interconnects as shown in Fig. 1(a) and (b) respectively. In addition to glitches and delays, presence of significant coupling inductance can result in damped voltage oscillations superimposed on top of a glitch or delay, as illustrated in Fig. 1(c). However, if the damping is large enough, the effects of the third case may be approximated by one of the first two cases. Also, previous studies [5, 12] have shown that crosstalk noise is more pronounced for long interconnects. Current and future SoCs will be dominated by a large number of very long interconnects and buses needed for the integration and communication among the cores in the chip [14]. In this paper, therefore, we focus on developing a comprehensive test solution to address the test needs of both the logic cores as well as the buses and global interconnects of an SoC.

Crosstalk effects most adversely affect high performance circuits operating at GHz clock frequencies. At-speed testing is essential for testing such chips adequately since many crosstalk effects are not manifested at lower speeds. However, the gap between ASIC speeds and external testers' accuracy for timing signal resolution at ASIC pins is constantly growing [14]. Furthermore, test equipment having high speed, large pin count, large memory, and good timing accuracy can be prohibitively expensive. We, therefore, employ self-testing techniques to address the problem of testing for crosstalk in SoCs.

In recent years, the power consumption of digital systems during testing has become a major concern as it may increase significantly as compared to normal operational mode [19]. Also, empirical studies have shown that the power dissipation associated with long interconnects accounts for a significant fraction of the overall system power [10]. This power consumption is dominated by the increasing interwire cross-coupling capacitances in DSM technology [15]. The energy dissipated due to cross-coupling capacitances can vary depending on the type of transitions on the interconnects [15]. We, therefore, also focus on making our self-test scheme extremely power-efficient.

Bai et al. [2], have proposed inserting dedicated interconnect self-test structures in the SoC to generate vectors which have 100% crosstalk defect coverage. This scheme is based on the Maximal Aggressor Fault Model proposed by Cuviello et al. [5]. However, this method has a prohibitively high area overhead. To reduce this overhead, existing logic BIST structures, like linear feedback shift registers (LFSRs), could be reused to generate interconnect tests. But, LFSR vectors have very poor crosstalk defect coverage. In this paper, we address the issue of how to generate high quality interconnect crosstalk tests, efficiently reusing existing on-chip test structures so as to minimize the area overhead. Our proposed methodology, called *Logic-Interconnect BIST* (LI-BIST), produces high crosstalk defect coverage with low area penalty and low interconnect power consumption.

The interconnect fault model used in this paper is the Maximal Aggressor Fault Model that was reported and validated in [5]. Next, we briefly review this fault model, and the corresponding Maximal Aggressor test vectors.

1.1. Interconnect Crosstalk Fault Model

If the crosstalk problem is addressed at the process level, the number of possible process variations and physical defects that need to be considered even for a pair of interconnects is very large. For wide buses, considering all such variations is clearly prohibitive. At the circuit level, the cumulative effect of process variations can be described behaviorally by a coarser mesh of lumped circuit elements, but the resulting fault universe is still too large. Hence, we need an abstract fault model that can represent all crosstalk defects with a small number of faults.

The Maximal Aggressor Fault Model (MAFM) [5] is a functional fault model representing all the process variations and physical defects that lead to any of the following four crosstalk errors on a wire designated as the victim wire among the set of interconnects under test: positive glitch (g_p), negative glitch (g_n), rising delay (d_r) and falling delay (d_f). All the other wires are designated as aggressors and act collectively to generate the glitch or delay error on the victim. Fig. 2 shows the transitions needed on the aggressors

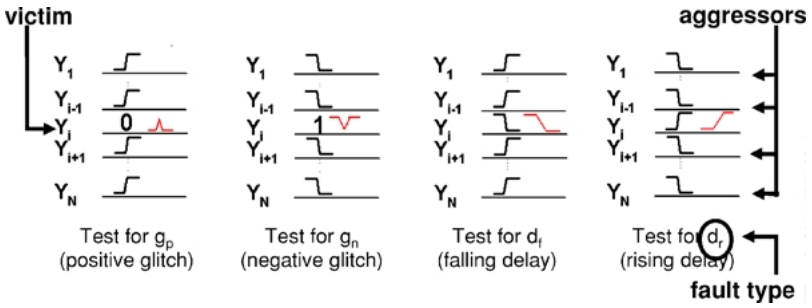

Fig. 2. Maximal aggressor fault model.

and victim wires to produce the maximum error effect for all four error types on the victim wire Y_i. These transitions constitute the Maximal Aggressor (MA) tests; they are necessary and sufficient for detecting the corresponding four crosstalk faults for the victim wire Y_i.

From Fig. 2 we see that the MA vectors are such that all the aggressor wires have a same direction transition while the victim wire remains at $0/1$ or has an opposite direction transition. Such vectors cause the maximum cross-coupling capacitive effect on the victim wire. Hence, vectors which cause a majority of the wires to transition in the same direction and relatively few wires to remain at $0/1$ or transition in the opposite direction (and thus exciting high cross-coupling capacitive effect on the victim wires) achieve high interconnect crosstalk defect coverage.

1.2. Paper Outline

In Section 2, we briefly describe the interconnect test scheme proposed by Bai et al. in [2]. In Section 3, we discuss how existing logic BIST structures could be reused to deliver interconnect tests, and the shortcomings thereof. These shortcomings are addressed in Section 4, where we propose a new test generator design which can generate vectors for both logic as well as interconnects and also present the overall test architecture for LI-BIST. Experimental results are presented in Section 5. Section 6 concludes the paper.

2. Interconnect Test Using Dedicated Self-Test Structures

A self-test methodology for testing interconnects based on the Maximal Aggressor Fault Model has been proposed by Bai et al. [2]. Here, we briefly describe their scheme.

The scheme is based on the fact that since the required Maximal Aggressor tests are known a priori, if suitable self-test structures can be inserted in the SoC to generate all the required MA vectors, then the self-test methodology will be able to achieve 100% crosstalk defect coverage. For each core to core test transaction, the methodology requires a test generator in the interconnect interface of the source core and an error detector in the interconnect interface of the destination core. For example, in the SoC shown in Fig. 3, to test the transaction from core *C1* (CPU) to core *C3* (RAM), a test generator is inserted at the output of the CPU core, and an error detector is inserted at the inputs of the RAM core. The test vectors are launched on the interconnect under test by the test generator of the source core and measured for logical consistency at the other end of the interconnect by the error detector in the destination core. Since the drivers and loads of the core play a crucial role in crosstalk noise, the test generators/error detectors are located between the core outputs/inputs and the core's buffer connections to the bus. A global test controller, which selects and activates appropriate test generators/error detectors is also described in the paper.

The total hardware overhead of this scheme applied to a Digital Signal Processing chip, CMUDSP, as reported in the paper, is about 22% which is clearly prohibitive. Hence, although this scheme achieves 100% crosstalk defect coverage, yet it is infeasible due to the high area overhead involved.

It should be noted that the proposed self-test structures are used only to test the interconnects of the SoC. They are exclusive of any other test structures that may be present on-chip to test the logic of the SoC. If existing logic BIST structures could be efficiently reused to deliver interconnect tests, then the need for separate interconnect test generators/error detectors can be done away with and the area overhead kept at a

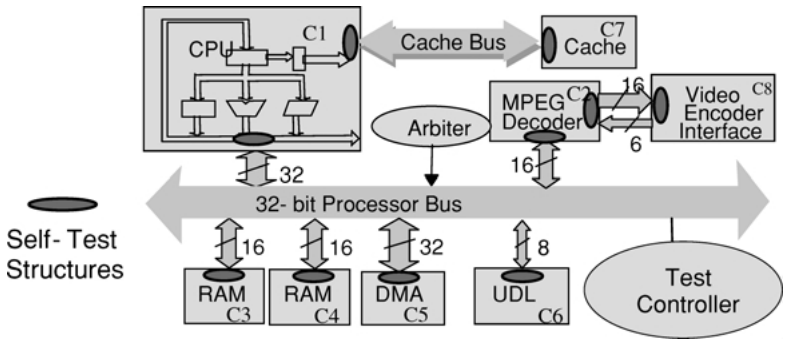

Fig. 3. An SoC with embedded self-test structures.

minimum. We describe such a scheme in the next section.

3. Reuse of Logic BIST for Crosstalk Testing

In this section, we describe how existing logic BIST structures could be reused to deliver interconnect tests, and the shortcomings thereof. In the next section, we propose our new test generator design which addresses these shortcomings.

Consider an SoC in which logic BIST structures, comprising of Linear Feedback Shift Registers (LFSRs) and Multiple Input Signature Registers (MISRs), are used to test the logic of the circuit. Since in an SoC, each core is itself a large and complex circuit, all cores are assumed to have their own dedicated LFSR/MISR. In our proposed methodology, the LFSRs will be used to generate test vectors not only for the logic cores but also for the interconnects. Similarly, the MISRs will be used for compacting the output responses of both the logic cores as well as the interconnects. There are two test phases, the logic test phase and the interconnect test phase. During the logic test phase, the cores are tested using their own LFSRs/MISRs as in traditional logic BIST. During the interconnect test phase, for each core-to-core test transaction, the LFSR of the source core generates the test vectors, which are then delivered onto the bus and they are compressed by the MISR at the destination core for signature analysis.

This scheme is illustrated by Fig. 4. The example SoC contains two cores *C1* and *C2* that communicate using interconnects *I1* and *I2*. Each core has a dedicated LFSR/MISR for logic BIST (*L1*/*M1* for core *C1* and *L2*/*M2* for core *C2*). The figure shows an example logic BIST configuration. The core flip-flops are scanned and the I/Os are boundary scanned. There is a multiplexer at each core (*MUX1* and *MUX2*) to choose

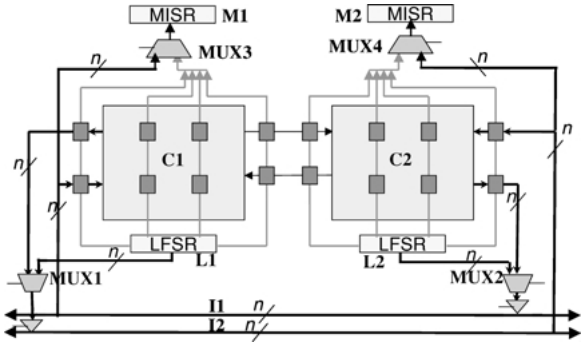

Fig. 4. Testing for crosstalk defects on interconnects reusing logic BIST structures.

whether the values going on the interconnects are test vectors generated by the LFSR or normal operational values from the core. Similarly there is a multiplexer (*MUX3* and *MUX4*) to choose whether the MISR compresses the output responses of the logic core or the values on the interconnects. Suppose we are testing for crosstalk defects on *I1*, for the core-to-core transaction $C2 \rightarrow C1$. The vectors generated by *L2* (the source core's LFSR) are driven on *I1*, and are compacted by *M1* (the destination core's MISR). At the end of the test, MISR *M1*'s signature is analyzed to see if there was any crosstalk error for this particular transaction. Similarly, for testing the interconnect *I2*, for the core-to-core transaction $C1 \rightarrow C2$, the vectors are generated by *L1* and compacted by *M2*. Note that the interconnect test scheme does not depend upon the actual logic BIST configuration as long as an LFSR and MISR are present at each core.

This scheme looks attractive since the hardware overhead incurred is very minimal. It essentially reuses existing logic BIST structures to act as test generators and error detectors for testing crosstalk defects in interconnects. But, unfortunately the vectors

generated by an LFSR, though good for logic, have poor crosstalk defect coverage [17]. This is because the kind of vectors which achieve high stuck-at-fault coverage for logic are very different from those which achieve high interconnect crosstalk defect coverage. As shown in Section 1.1, good vectors for interconnect crosstalk test are such that they cause a majority of the wires to transition in the same direction and a few wires to remain at 0/1 or transition in the opposite direction, thus exciting high cross-coupling capacitive effect on the victim wires. The vectors produced by an LFSR are pseudo-random and do not have such a characteristic. Hence, they have poor defect coverage.

We would ideally like to use this framework of testing the interconnects (as it has low area overhead) but generate test vectors having much higher crosstalk defect coverage. Thus, our goal is to develop a test generator, which while not compromising on logic fault coverage, produces high quality interconnect crosstalk tests with minimal area overhead. We call this integrated self-test scheme for both logic and interconnects as LI-BIST which we present in the next section.

4. New Test Generator Design and LI-BIST Test Architecture

In this section, we first present the design of the test generator for LI-BIST and discuss the motivation behind such a design. We then present the test architecture of LI-BIST.

4.1. New Test Generator Design

Before presenting the actual design of the test generator, we first discuss the desired properties of LI-BIST and the corresponding requirements of the test generator:

- *Low Cost*: The test generator should be designed so as to maximize the reuse of existing self-test structures (logic BIST) so that additional test circuitry area is minimized.
- *High Logic and Interconnect Defect Coverage*: The new test generator should achieve as high logic fault coverage as existing logic BIST schemes. Also, it should produce high quality interconnect crosstalk tests. Hence, the profile of the interconnect test vectors produced by the test generator should be such that the majority of the wires transition in the same direction (to act as aggressors) and relatively few wires remain 0/1 or transition in the opposite direction (to act as victims) so as to cause a high cross-coupling capacitive effect on the victim wires, as described in Section 1.1.
- *Low Power*: The vectors generated by the test generator should be interconnect power efficient. Hence, we should minimize opposite direction transitions on adjacent wires, so that the *total amount* of coupling capacitance excited is low.

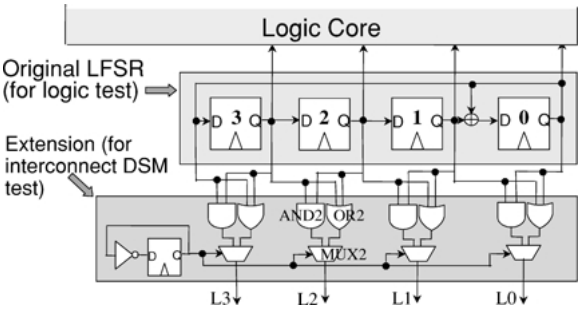

Fig. 5. A 4 bit test generator for LI-BIST.

Fig. 5 shows the structure of a 4 bit LI-BIST test generator. It consists of an LFSR that produces vectors for testing the logic core (as in traditional logic BIST), and an extension circuit that modifies the LFSR vectors such that they have high interconnect crosstalk defect coverage. In the extension circuit, the adjacent bits of the LFSR are both *ANDed* and *ORed*. For example, LFSR bits 2 and 3 are input to *AND2* and *OR2*. The outputs of the AND gate and the OR gate are connected to a 2:1 multiplexer. The multiplexer selects which of these values drives the corresponding output line. For example, *MUX2* selects whether the output line *L2* is driven by the output of *AND2* or *OR2*. The select line of all the multiplexers is driven by a toggle flip-flop. Hence, the value of an output line alternates at every cycle between the output of the respective *AND* gate and *OR* gate. For example, at clock cycle i, if the *AND* gates drive the lines *L0* to *L3*, then the *OR* gates drive them at cycle $i + 1$, the *AND* gates drive them at cycle $i + 2$, and so on.

With this small extension to a regular LFSR, the interconnect test vectors generated have the required profile as described earlier in this section. For an LFSR configured with a primitive polynomial, the probability of any bit i being 0, $P_i(0)$, is $1/2$; $P_i(1) = 1/2$. So the probability that the *AND* of two bits, i and j, is 0 is $3/4\,(1 - P_i(1) * P_j(1))$. Similarly, the probability that

the *OR* of two bits, i and j, is 0 is $1/4\,(P_i(0) * P_j(0))$. Now, since the interconnect vectors generated are driven alternately by the *AND* gates and the *OR* gates, hence the probability that an output line is 0 varies alternately between 3/4 and 1/4. So, we are essentially generating weighted random patterns which are alternately weighted with a high probability of 0 and a high probability of 1. Hence, the profile of the interconnect test vectors generated will be such that many wires transition in the same direction to act as aggressors, and a few wires remain static or transition in the opposite direction to act as victims; exactly the kind of vectors we require. Note that the interconnect test vectors are not *true* MA vectors but *MA like* vectors, since there might be multiple victims. However, only the adjacent few aggressor wires on either side of the victim have any significant cross-coupling capacitive effect on the victim wire, and so are sufficient to excite the fault.

In order to minimize the correlation among the generated interconnect vectors due to the structural dependencies among the LFSR bits, ideally a separate LFSR should be used for each output line, all of them differently configured. However, such a scheme is expensive in terms of area. Our proposed scheme using only a single LFSR to generate all the bits, is more than adequate for our requirements since we do not really care what the exact probabilities are as long as alternate vectors have a high probability of 0 and 1. For the same reason, the seed value of the LFSR and the primitive polynomial used does not affect the crosstalk defect coverage much.

The size of the extension circuit which is added to the LFSR to appropriately weight the interconnect test vectors is relatively small. For testing an n bit bus, for example, we require n 2-input *AND* gates, n 2-input *OR* gates, n 2-bit *MUX* gates, **1** *DFF* and 1 *NOT* gate.

4.2. *Test Architecture for LI-BIST*

Fig. 6 illustrates the test architecture of LI-BIST on an example SoC consisting of 2 cores. Each core is surrounded by a *Test Wrapper*, which consists of self-test structures like a Test Pattern Generator (TPG) (described in the previous section) and a MISR. The TPG generates test vectors for both the logic core as well as the interconnects. Multiplexers are used to select between the normal core outputs and the interconnect test vectors. The MISR compacts the output responses of both the core and the interconnects; this is

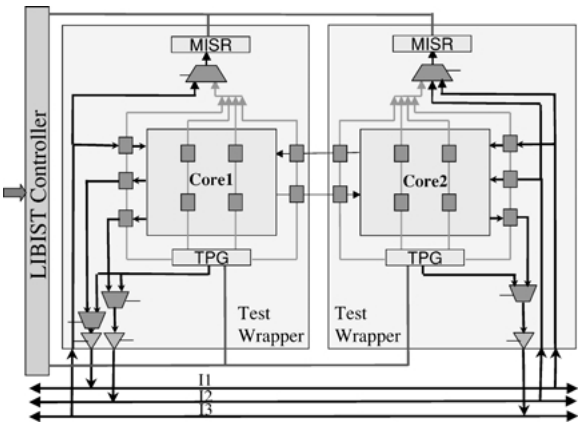

Fig. 6. Test architecture of LI-BIST.

selected by multiplexers. There is a centralized *LI-BIST Controller* which controls the whole test sequence. It gives control signals to all the test structures and is also responsible for seeding the TPG and unloading the MISR signature.

There are three operational modes of the SoC; normal mode, core test mode and interconnect test mode. In the normal mode, the SoC functions in its normal system operational mode. During the core test mode, the internal logic of the cores is tested via the TPG and the MISR, as in logic BIST. The core F/Fs are scanned and the I/Os are boundary-scanned for controllability and observability. In the interconnect test mode, for each core-to-core transaction, the vectors generated by the TPG of the source core are routed onto the interconnect and they are compacted by the MISR of the destination core. For example in Fig. 6, there are three core-to-core transactions which are tested: $Core1 \xrightarrow{I_1} Core2$, $Core1 \xrightarrow{I_2} Core2$, $Core2 \xrightarrow{I_3} Core1$.

The LI-BIST test scheme can be made to work with the emerging IEEE P1500 Standard for Embedded Core Test (SECT) [8]. Since the mechanism for testing the logic cores is the same, hence the LI-BIST test wrapper can be merged with the P1500 wrapper. The LI-BIST TPG is used as the source of vectors and the MISR is used as the sink. The LI-BIST controller can be merged with the P1500 controller to control the whole test sequence.

5. Validation of LI-BIST Methodology

In order to validate the LI-BIST methodology, we applied it to two example SoCs: (i) a sub-system of

the 802.11 MAC layer [16], and (ii) a Digital Signal Processing chip, CMUDSP [3], which corresponds to Motorola DSP56002. In this section, we first briefly describe the architecture of these SoCs and discuss how LI-BIST is applied to these chips. Next, we describe how the interconnect crosstalk defect coverage of LI-BIST is measured using a high-level crosstalk defect simulation methodology. Finally, we present our experimental results by comparing LI-BIST with the previous two schemes outlined in Sections 2 and 3 w.r.t the MAC sub-system and the CMUDSP chip.

5.1. *Examples*

In this subsection, we describe the architecture of the sub-system of the 802.11 MAC layer and the CMUDSP chip and discuss how LI_BIST is applied to these chips.

5.1.1. Case Study 1: A Sub-System of the 802.11 MAC Layer. We first applied the LI-BIST scheme on a sub-system of the 802.11 MAC layer. Fig. 7 shows the architecture of this sub-system, which consists of the PARWAN processor core [11] and a hardware encryption co-processor. PARWAN is a simple accumulator-based microprocessor. The data bus is 8-bits wide and shared by both *Data In* and *Data Out*. The address bus is 12-bits wide. The *Read* and *Write* signals indicate whether data is read from or written to the address specified.

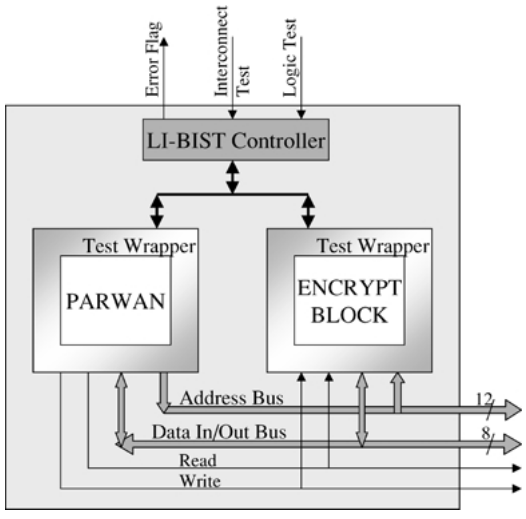

Fig. 7. Application of LI-BIST to a sub-system of the 802.11 MAC layer.

The encryption co-processor implements the 802.11b WEP encryption algorithm [16]. PARWAN controls the encryption process by writing to the encryption co-processor registers which are memory-mapped.

In order to apply LI-BIST, both the PARWAN and the encryption co-processor are enclosed in a test-wrapper as described in Section 4.2. The test wrapper contains test structures for testing both the logic cores as well as the interconnects. An LI-BIST controller is also inserted, which interfaces with the test wrappers by means of control and data signals, as shown in Fig. 7. The LI-BIST controller initiates logic or interconnect tests when it receives corresponding external signals. At the end of the tests, it signals whether the chip is good or not through the error flag.

5.1.2. Case Study 2: CMUDSP. Fig. 8 shows the CMUDSP chip, which consists of four components: Arithmetic and Data Logic Unit (ALU), Address Generation Unit (AGU), Bus Switch, and Program Control Unit (PCU). The ALU contains the X, Y, A and B registers along with the multiply-accumulate and adder units. The AGU generates the addresses for accessing the data memories. The PCU contains the program counter and flag bits for controlling the whole CMUDSP chip. The PCU also performs program address generation and instruction decoding. The Bus Switch is used to control data flow between buses. CMUDSP consists of four sets of separate data buses (XDB, YDB, PDB, GDB) and address buses (XAB, YAB, PAB). The X and Y buses are connected to the data memory, and the P buses are connected to the program memory.

As described in the previous sub-section, in order to apply LI-BIST, each of the components is wrapped in a test wrapper and a centralized LI-BIST controller is inserted which controls the whole test sequence as shown in Fig. 8.

5.2. *Fault Simulation Methodology*

In order to calculate the interconnect crosstalk defect coverage of LI-BIST for any SoC, we have to inject and simulate crosstalk defects in the interconnects, and examine how many of these defects are detected by the interconnect test vectors. The most accurate way of doing this is spice-level simulation. However, it is not feasible to simulate the entire chip at this level since it takes prohibitively long. Hence, we use a high-level crosstalk defect simulation method [1], which allows

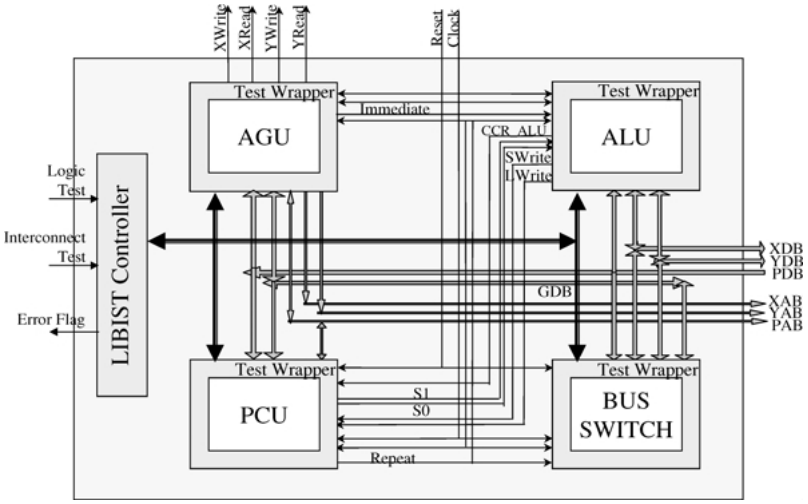

Fig. 8. Application of LI-BIST to CMUDSP.

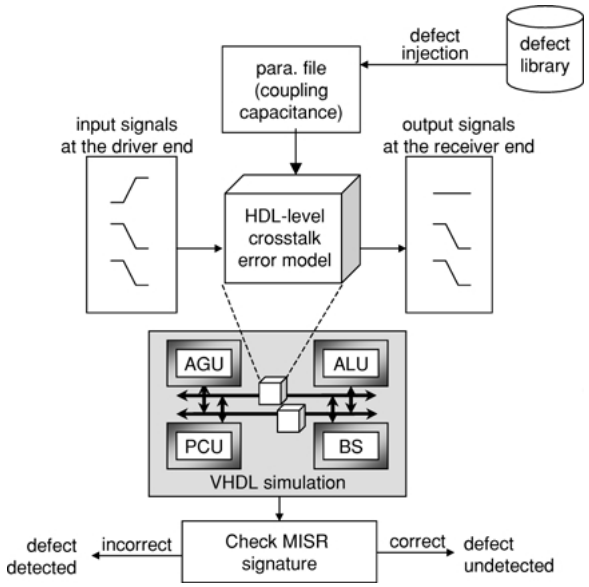

Fig. 9. Crosstalk defect simulation environment.

simulation of the crosstalk effects on the interconnects, together with the HDL models of the rest of the chip components including the test structures. We describe this fault-simulation methodology briefly here using the CMUDSP chip as an example.

The high-level crosstalk defect simulation environment for the CMUDSP chip is shown in Fig. 9. Behavioral level interconnect DSM error models [1] are inserted corresponding to the interconnects under test so that we can inject crosstalk defects in the interconnects and simulate the behavior of defect effects (like glitches and delays) at the high-level. Coded in HDL, the error model takes as input a parameter file containing the values of the coupling capacitances among the interconnects. Given an input transition on the driver end of the bus, the error model determines whether a crosstalk error occurs at the receiver end, based on error criteria described in [5]. The criterion for a glitch or delay error depends on the technology and the characteristics of the receiving logic like setup time, hold time, noise threshold voltage etc [5]. A crosstalk defect can be injected by perturbing the values and the distribution of the cross-coupling capacitances in the parameter file beyond a threshold value to reflect process variation beyond design margins.

Fault-simulation is performed as follows: An HDL test generator of the source core generates the test vectors for a bus, which are input to the corresponding DSM error model. Depending upon the test vector transitions and the coupling parameters of the bus from the parameter file, the model generates output vectors, which may or may not contain digital errors corresponding to glitches or delays. Then, the MISR at the destination core compresses these output vectors. At the end of the test, the MISR signature is analyzed to see whether the injected defect was detected or not. To estimate the defect coverage, the same defect simulation process is repeated on all defects from a preconstructed defect library.

To generate the defect library, we randomly perturb the nominal values of coupling capacitances among the

interconnects according to a given defect distribution. Given the resulting perturbation, we use the error criteria in [5], to determine if the perturbation is large enough to be detected by *any* tests. If so, we record the perturbation as a defect. This process is repeated until a satisfactory number of defects are generated.

In our experiments, we used a Gaussian distribution to model the defect distribution in terms of the variation of capacitance values (in %). A standard variance of 50% was chosen. A total of 1000 defects were generated for each bus.

5.3. Experimental Results

In this subsection, we compare the LI-BIST scheme with the MA Test scheme described in Section 2 and the LFSR test scheme described in Section 3 in terms of the interconnect defect coverage, the area overhead, the interconnect power consumption during testing, and the test application time. The experiments were applied to the two SoCs, the sub-system of the 802.11 MAC layer and the CMUDSP chip described in Section 5.1.

5.3.1. Defect Coverage. Table 1 compares the three schemes in terms of the interconnect crosstalk defect coverage, and the logic fault coverage of the cores for the 802.11 MAC sub-system and the CMUDSP chip. All the three schemes were implemented for both the chips. For the 802.11 MAC sub-system, LFSRs of 24 bits and 12 bits were used for the PARWAN core and the ENCRYPT core respectively. LFSRs of 32 bits were used for the AGU, ALU and PCU cores and an LFSR of 16 bits was used for the BUS_SWITCH core in the CMUDSP chip. For both the chips, the test length for logic was 32757 vectors and the test length for the interconnects was 10000 vectors. The defect coverage was measured using the high-level fault-simulation methodology described in the previous sub-section. The logic fault coverage of the cores is in terms of single stuck-at-fault coverage and was measured using Mentor Graphics' FastScan [7].

Table 1. Defect coverage comparison of MA tests, LFSR tests and LI-BIST.

Design	Test scheme	Defect coverage (%)	Logic fault coverage (%)
802.11	MA tests	100	–
MAC	LFSR tests	3.4	96.71
sub-system	LI-BIST	100	96.71
CMUDSP	MA tests	100	–
	LFSR tests	1	91.36
	LI-BIST	99.7	91.36

From Table 1, we observe that the MA tests have 100% defect coverage because the interconnects are tested using MA vectors. The LFSR tests are not feasible due to their extremely poor defect coverage. The LI-BIST scheme, however, achieves very high defect coverage comparable to MA tests. Both LFSR tests and LI-BIST achieve the same logic fault coverage of the cores since the test methodology for logic is the same in both the schemes.

Table 2. Area overhead comparison of MA tests, LFSR tests and LI-BIST.

Design	Test scheme	Area overhead (%)
802.11	MA tests	20.05
MAC	LFSR tests	3.55
sub-system	LI-BIST	6.24
CMUDSP	MA tests	22
	LFSR tests	1.22
	LI-BIST	4

5.3.2. Area Overhead. Table 2 compares the three schemes in terms of the additional area overhead over conventional logic BIST for testing the interconnects for the 802.11 MAC sub-system and the CMUDSP chip. We used Synopsys' Design Compiler [6] synthesis tool to synthesize both the designs along with the test structures for all the three schemes.

From Table 2, we observe that the MA test scheme has high area overhead because it uses dedicated self-test structures for testing the interconnects. The LI-BIST scheme has very low area overhead since it reuses existing logic BIST test structures to test the interconnects also. Note that the LFSR test scheme has slightly more area compared to conventional logic BIST because of the extra multiplexers required to multiplex the LFSR and MISR signals (see Fig. 4).

5.3.3. Interconnect Power Consumption. In order to assess the power-efficiency of LI-BIST, we conducted another set of experiments to measure the average interconnect power consumption of all the three schemes. The results are presented in Table 3. The interconnect power consumption was measured using a

Table 3. Interconnect power consumption of MA tests, LFSR tests and LI-BIST.

Design	Test scheme	Avg. int. power (mW)
802.11	MA tests	0.39
MAC	LFSR tests	1.65
sub-system	LI-BIST	0.41
CMUDSP	MA tests	0.43
	LFSR tests	3.66
	LI-BIST	0.77

DSM-accurate power estimation technique proposed in [15] which takes into account not only the number of transitions on the interconnects but also DSM crosstalk effects.

The LFSR test vectors consume the maximum amount of interconnect power since the vectors are pseudo-random and they have many opposing direction transitions which excite a large *amount* of coupling capacitance. The MA vectors have the least power consumption because all the wires, except the victim, transition in the same direction. So, although there is a high coupling capacitive effect on the victim wire, the total amount of coupling capacitance excited is very low. The LI-BIST generated vectors have marginally higher interconnect power consumption as compared to the MA vectors since there may be more than one victim wire. So, the LI-BIST test methodology is also power-efficient.

5.3.4. Test Application Time. We conducted another set of experiments to compare the test application time of LI-BIST with the MA test scheme and the LFSR test scheme and to measure how the test application time varies with the bus width.

Fig. 10 shows the interconnect crosstalk defect coverage versus the number of test patterns applied for all the three schemes for a 24 bit bus. The interconnect defect coverage is measured using the high-level fault-simulation methodology described in Section 5.2. The MA test scheme very quickly reaches 100% defect coverage because all the test vectors are MA vectors and are deterministically generated. The LFSR test scheme has extremely poor defect coverage even after the application of a large number of patterns. The LI-BIST scheme quickly achieves high defect coverage with a small number of test patterns and then there is a marginal increase in the defect coverage with additional test patterns. This is because most of the

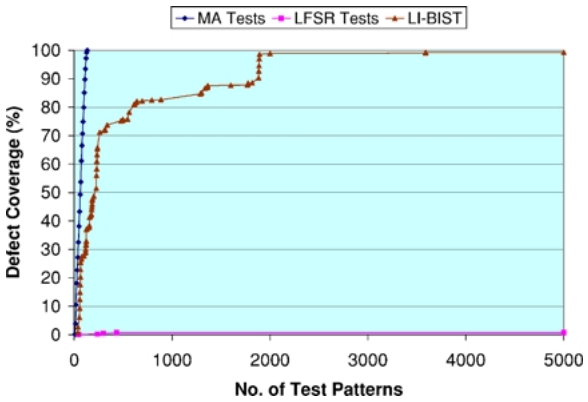

Fig. 10. Defect coverage v.s. test application time for MA tests, LFSR tests and LI-BIST for a 24 bit bus.

crosstalk faults can be detected by many different test vector pairs and so are detected early on. However, there are a few faults which can be detected only by a few specific test vector pairs and so additional test patterns are needed to detect them.

Fig. 11 shows how the test application time for the LI-BIST varies with the bus width for different defect coverage values. The LI-BIST vectors are generated by an LI-BIST test generator which consists of a 32 bit LFSR and the extension circuit as described in Section 4.1. From the figure, we can see that the LI-BIST scheme quickly achieves high crosstalk defect coverage with a small number of patterns for different bus widths. However, to achieve higher fault coverage, the additional number of LI-BIST patterns required increases with the bus-width. This is because as the bus width increases, the number of faults which can be detected by only a few specific test vectors

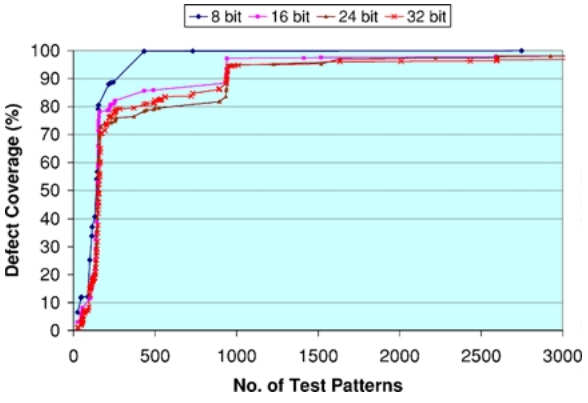

Fig. 11. Defect coverage v.s. test application time of LI-BIST for different bus widths.

also increases, and so more number of test vectors are required. The slight anomaly between the curve for the 24 bit bus and the 32 bit bus between the patterns 200 to 1000 can be attributed to the pseudo-random nature of the LI-BIST vectors.

6. Conclusion

In this paper, we presented LI-BIST, a comprehensive SoC test methodology for both logic cores and interconnects. It efficiently reuses existing on-chip test structures to generate high quality interconnect crosstalk tests. We have compared LI-BIST with existing solutions and validated it using a high-level crosstalk error model. Experiments on example SoCs confirm that LI-BIST yields high crosstalk defect coverage, low area overhead and low interconnect power consumption.

References

1. X. Bai and S. Dey, "High-Level Crosstalk Defect Simulation for System-on-Chip Interconnects," in *Proc. IEEE VLSI Test Symposium*," April 2001, pp. 169–175.
2. X. Bai, S. Dey, and J. Rajski, "Self-Test Methodology for at-Speed Test of Crosstalk in Chip Interconnects," in *Proc. Design Automation Conf*., June 2000, pp. 619–624.
3. The Carnegie Mellon Synthesizable Digital Signal Processor Core (http://www.ece.cmu.edu/low-power/benchmarks.html).
4. Z. Chen and I. Koren, "Crosstalk Minimization in Three-Layer HVH Channel Routing," in *Proc. IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems*, 1997, pp. 38–42.
5. M. Cuviello, S. Dey, X. Bai, and Y. Zhao, "Fault Modeling and Simulation for Crosstalk in System-on-Chip Interconnects," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1999, pp. 297–303.
6. Design Compiler 2000.05, Synopsys Inc. (http://www.synopsys.com).
7. FastScan v8.6_4.5 Mentor Graphics Corp. (http://www.mentor.com).
8. IEEE P1500 Standard for Embedded Core Test (http://grouper.ieee.org/groups/1500).
9. H. Kawaguchi and T. Sakurai, "Delay and Noise Formulas for Capacitively Coupled Distributed RC Lines," in *Proc. Asian and South Pacific Design Automation Conference*, 1998, pp. 35–43.
10. R. Mehra, L.M. Guerra, and J.M. Rabaey, "A Partitioning Scheme for Optimizing Interconnect Power," *IEEE J. Solid-State Circuits*, vol. 32, pp. 433–443, March 1997.
11. Z. Navabi, *VHDL: Analysis and Modeling of Digital Systems*, New York: McGraw-Hill, 1993.
12. P. Nordholz, D. Treytnar, J. Otterstedt, H. Grabinski, D. Niggemeyer, and T.W. Williams, "Signal Integrity Problems in Deep Submicron Arising from Interconnects Between Cores," in *Proc. IEEE VLSI Test Symposium*, April 1998, pp. 28–33.
13. K. Rahmat, J. Neves, and J. Lee, "Methods for Calculating Coupling Noise in Early Design: A Comparative Analysis," in *Proc. Int. Conf. Computer Design VLSI in Computers and Processors*, 1998, pp. 76–81.
14. Semiconductor Industry Association, "International Technology Roadmap for Semiconductors," 1999.
15. C.N. Taylor, S. Dey, and Y. Zhao, "Modeling and Minimization of Interconnect Energy Dissipation in Nanometer Technologies," in *Proc. Design Automation Conf.*, June 2001, pp. 754–757.
16. "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specications," IEEE Computer Society LAN/MAN Standards Committee, IEEE Std 802.11-1999 Edition.
17. Y. Zhao and S. Dey, "Analysis of Interconnect Crosstalk Defect Coverage of Test Sets," in *Proc. Int. Test Conf.*, Oct. 2000, pp. 492–501.
18. H. Zhou and D.F. Wang, "Global Routing with Crosstalk Constraints," in *Proc. Design Automation Conf*., 1998, pp. 374–377.
19. Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," in *Proc. IEEE VLSI Test Symposium*, April 1993, pp. 4–9.

Krishna Sekar is a Ph.D. candidate in the Electrical and Computer Engineering Department at the University of California, San Diego. His research interests include low-power system design, reconfigurable systems and self-test of system-on-chips. Sekar has a B.Tech. in Computer Science and Engineering from the Indian Institute of Technology, Kharagpur, India; and a MS in Electrical and Computer Engineering from the University of California, San Diego. He is a student member of the IEEE.

Sujit Dey is a Professor in the Electrical and Computer Engineering Department at the University of California, San Diego. His research group at UCSD is developing configurable platforms, consisting of adaptive wireless protocols and algorithms, and deep submicron adaptive system-on-chips, for next-generation wireless appliances and network infrastructure devices. He is affiliated with the California Institute of Telecommunications and Information Technology, the UCSD Center for Wireless Communications, and the DARPA/MARCO Gigascale Silicon Research Center. Prior to joining UCSD in 1997, he was a Senior Research Staff Member at the NEC C&C Research Laboratories, Princeton, NJ. He obtained a Ph.D. in Computer Science from Duke University in 1991. He has coauthored more than 100 publications, and is a co-inventor of 9 US patents, with several pending. He has received several Best Paper awards at conferences like the Design Automation Conference and the VLSI Design Conference. He has been the General Chair and Program Chair, and member of organizing and program committees, of several IEEE conferences and workshops.