Software-Based Self-Test Methodology for Crosstalk Faults in Processors *

Xiaoliang Bai, Li Chen and Sujit Dey

Department of Electrical and Computer Engineering
University of California, San Diego
{xibai, lichen, dey}@ece.ucsd.edu

Abstract

Due to signal integrity problem's inherent sensitivity to timing, power supply voltage and temperature, it is desirable to test AC failures such as crosstalk-induced errors at operational speed and in the circuit's natural operational environment. To overcome the daunting cost and increasing performance hindrance of high-speed external testers, Software-Based Self-Test (SBST) is proposed as a high-quality low-cost at-speed testing solution for AC failures in programmable processors and System-on-Chips (SoC). SBST utilizes low-cost testers, applies tests and captures test responses in the natural operational environment. Hence SBST avoids artificial testing environment and external tester induced inaccuracies.

Different from testing for stuck-at faults, testing for crosstalk faults requires a sequence of test vectors delivered at the operational speed. SBST applies tests in functional mode using instructions. Different instructions impose different controllability and observability constraints on a module-under-test (MUT). The complexity of searching for an appropriate sequence of instructions and operands becomes prohibitively high. In this paper, we propose a novel methodology to conquer the complexity challenge by efficiently combining structural test generation technique with instruction-level constrains. MUT in several time frames is automatically flattened and augmented with Super Virtual Constraint Circuits (SuperVCCs), which guide an automatic test pattern generation (ATPG) tool to select appropriate test instructions and operands. The proposed methodology enables automatic test program generation and high-fidelity test solution for AC failures. Experimental results are shown on a commercial embedded processor (Xtensa™ from Tensilica Inc.).

1. INTRODUCTION

Due to the overwhelming complexity of modern processors and significant process variations in advanced manufacturing technologies, testing for high-performance chips becomes expensive and time-consuming. For deep submicron circuits below 0.18µm, signal integrity (SI) problems are becoming ever more serious. SI problems can severely affect circuit performance, even cause timing and functional errors. Although, to some extend, noise analysis and repair-by-construction help to suppress SI problems, random defects and excessive process variations can cause unexpected AC failures. Hence SI problems have to be considered in manufacturing test.

Various test methodologies such as scan test, built-in self-test (BIST), and functional test have been adopted to ensure the quality of manufactured chips. Scan test can be systematically applied to complex designs. It enables automatic test generation and enhances diagnostic resolution. BIST generates at-speed tests and helps to relieve the requirement of high-speed external testers. Structural tests like scan and BIST provide sufficient, scalable solutions for traditional stuck-at faults. As a result, they are heavily exploited.

Both scan (such as enhanced scan [1]) and BIST can be used to test signal integrity problems. However, they have several limitations. Firstly, switching activities in structural tests are quite different from those in functional mode [2]. The abnormal large power dissipations in testing mode can cause permanent damage to the chip [3]. More importantly, signal integrity problems are strongly sensitive to supply voltage, temperature, and timing. Structural tests apply vectors in an "artificial" environment. Hence, the voltage, temperature and timing are different from operational mode for the module-under-test (MUT) and surrounding logic. Consequently, circuit behavior dramatically changes [4]. Thirdly, the vectors generated in structural tests may never occur in functional mode, thus a "detected" failure may never happen in the field. All these limitations may lead to over-test or under-test for the manufactured chips.

Contrary to structural tests, functional tests exercise MUTs in the chip's natural operational environment. Functional tests can be performed at operational speed without introducing area and delay overhead. Hence, for speed binning and AC tests on high performance processors and SoCs, functional test is indispensable.

In spite of these attractive benefits, several challenges frequently hinder the application of functional tests. It's well known that high-speed testers are prohibitively expensive and the performance gap between automated test equipments (ATE) and device I/O speed is increasing. Timing inaccuracies of external testers are becoming comparable to the cycle time of the fastest devices on chip

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

[5]. Moreover, functional test generation is time-consuming and labor-intensive. Many tests are migrated from design verification suite without targeting any physical defects and need to be augmented with manually developed tests. Furthermore, testing for signal integrity problem requires layout information that is available in late design phase, resulting in a very pressing time budget for test generation.

Software-Based Self-Test (SBST) has been proposed to alleviate the need for high-speed testers. Utilizing low-speed testers, SBST uses on-chip programmable resources such as processors to generate tests and compress test results. To ease the burden of functional test generation, methods of using random instruction sequences have been developed [6][7][8]. Since such test generation processes are not defect-oriented, to achieve high defect coverage, it requires a long test program with increased fault simulation time and tester time. Deterministic test methods have been developed to mitigate this problem [9][10]. Based on a divide-and-conquer approach, SBST first generates tests for a specific MUT. Processor instructions are then used as a vehicle to deliver test patterns and collect test responses.

Unlike scan test that possesses enhanced controllability and observability on internal modules, for SBST, many module-level ATPG vectors cannot be delivered to the MUT. The instruction set architecture (ISA) and surrounding logic impose various requirements on the input vectors and the output observable nodes. These special requirements determine what kind of vectors can be applied by processor instructions, and are called ***instruction-imposed constraints***. To guarantee the delivery of module-level ATPG vectors by instructions, these constraints should be considered during test generation.

Automated methods for extracting surrounding-logic-imposed constraints were proposed in [11][12]. Since they are only a subset of instruction-imposed constraints, these constraints cannot be directly applied to SBST. Symbolic simulation was used to extract constraints imposed by ISA [13]. Using these constraints and targeting path delay faults, an automated test program synthesis method was developed based on a backtrack-based searching algorithm [14]. However, scalability remains a big challenge. In [15], a systematic SBST methodology was proposed using a simulation-based constraint extraction method and a constrained ATPG method, enabling automatic test program generation for stuck-at faults in industrial-size processors.

Challenges in Testing Crosstalk Faults

Dissimilar to stuck-at faults, signal integrity problems such as crosstalk need to be tested by applying ***a sequence of vectors*** at operational speed. The requirements of generating consecutive multiple vectors and considering instruction-imposed constraints pose difficult challenges on test program generation.

To detect crosstalk-induced errors, a sequence of appropriate instructions is needed to deliver test vectors at operational speed. Different instruction combinations impose different constraints and hence determine the potential test quality that can be delivered. In the process of test generation for stuck-at faults, a fault can be dropped from the fault list whenever it is detected by a legal instruction. Contrasting to test generation for stuck-at fault, test generation for crosstalk fault is much more complicated. All possible instruction sequences have to be examined for the possibility of delivering test vectors and compared to find the one that can potentially trigger the largest noise. For modern processor, the ISA can have hundreds of instructions. The number of instruction sequences need to be considered is huge. Test program generation method utilizing test-instruction templates [15] becomes infeasible for crosstalk testing.

Paper Outline and Contributions

In this paper, we propose a methodology that enables efficient defect-oriented test program generation for crosstalk faults using SBST. We develop a novel non-enumeration technique that can tremendously reduce the complexity of the test program generation process, therefore renders SBST a scalable solution for faults requiring a sequence of vectors. Combining instruction-level constraints with structural ATPG algorithms through the proposed non-enumeration technique, the selection of instruction sequences and corresponding operand values can be accomplished effectively, enabling automated test program generation with little human intervention. Pipeline forwarding and data correlation among multiple instructions are also considered. An automated test program generation framework is developed and applied to a state-of-the-art commercial processor (Xtensa™ from Tensilica Inc. [16]).

The rest of the paper is organized as follows. Section 2 discusses challenges in applying SBST for crosstalk faults in details. Section 3 outlines our methodology and describes key steps to reduce the complexity of constrained ATPG in multiple timeframes and an efficient conflict-aware test program generation technique. Experimental results on the Xtensa processor are presented in section 4. Section 5 concludes the paper.

2. TESTING FOR CROSSTALK FAULTS

As a major source of signal integrity problem, crosstalk noise can cause functional and timing errors. A wire, whose signal is under concern, is a *victim*. An *aggressor* is a net whose signal transitions can potentially deteriorate the victim's signal because of a large coupling capacitance or a weak and sensitive victim wire.

Various crosstalk noise effects can cause failures in the circuit-under-test (CUT). Due to random defects and excessive process variations, a large noise can cause logic

and timing failures. In this work, we focus on crosstalk-induced logic and transition faults, which include positive glitch, negative glitch, rising delay, falling delay, rising speedup and falling speedup, as shown in Figure 1. However, the methodology that will be elaborated can also be applied to other AC fault models, such as path delay faults.

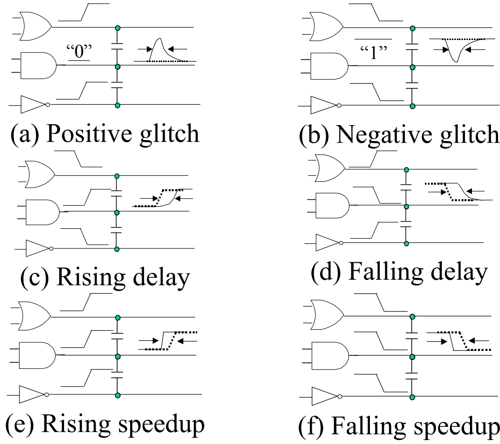


Figure 1: Crosstalk faults

For a given crosstalk fault, test program launches test vectors to trigger a set of aggressors, sensitizes a propagation path and stores the test results to the memory. It is a multiple-timeframe test generation problem. Like testing for stuck-at faults in SBST, vectors that can be delivered by instructions are limited. In order to be able to translate ATPG results into legal instructions, module-level ATPG should be guided by instruction-imposed constraints. Next, we briefly recap the techniques that enable automatic test program generation for stuck-at faults.

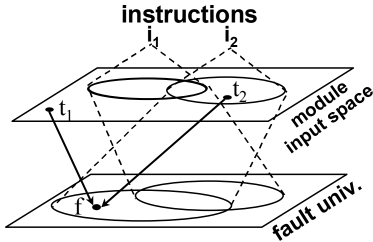


Figure 2: Instruction-impose constraints on input space

In SBST, test vectors are delivered to the MUT in functional mode by instructions. Vectors that can be delivered by each instruction form a subset of the original input space. Since not all vectors can be generated in functional mode, the superset of all the instruction-related subsets composes a constrained input space. Correspondingly this reduced input space maps to a subset of the fault universe, as shown in Figure 2. Faults belonging to this fault subset are those detectable by functional vectors. For a functionally detectable fault (such as fault *f*), ATPG tool should be guided to find a "*legal"* functional vector (t_2) that can be delivered by instructions, instead of an "illegal" vector (t_1).

In [15], an enumeration-based methodology was developed that enables automatic test program generation for stuck-at faults. Constraints were firstly extracted for each instruction. In [17], Tupuri *et al.* introduced the concept of virtual constraint circuit (VCC) that was later extended and adapted for SBST in [15]. The VCCs enforce instruction-imposed constraints on the MUT, providing constraint information in the form of logic circuits. Controllability VCCs (CVCCs) and observability VCCs (OVCCs) are attached to the inputs and outputs of the MUT, respectively. Constrained test generation is performed on a CUT containing the MUT and the VCCs. The primary inputs (PI) of controllability VCCs are mapped to *settable fields* (e.g. operands, source and destination registers) that are controllable in instructions. The primary outputs (PO) of observability VCCs are fields that can be mapped and stored to the memory [15]. Test generation is performed by automatically enumerating constraints for each instruction in the ISA. This methodology provides an efficient, scalable test program generation solution for stuck-at faults.

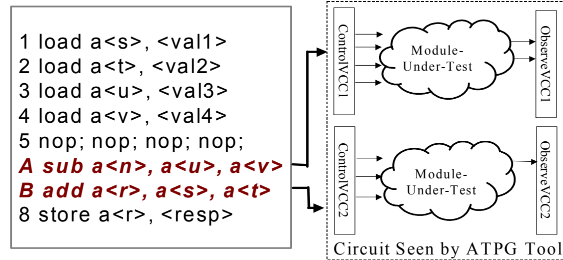


Figure 3: Test instructions and MUTs with VCCs

However, in testing for crosstalk fault, this method meets severe challenges. To detect a crosstalk fault, a sequence of test vectors needs to be delivered by multiple instructions. Figure 3 shows a test program that uses two instructions (instruction A and B) to deliver test vectors. Other peripheral instructions are used to setup appropriate operands for these two test-delivery instructions. Since instructions from different timeframes impose different constraints, the MUT will be augmented with different input/output VCCs. ATPG tool has to handle a problem that keeps changing across timeframes. Hence, it is more difficult than a traditional ATPG problem.

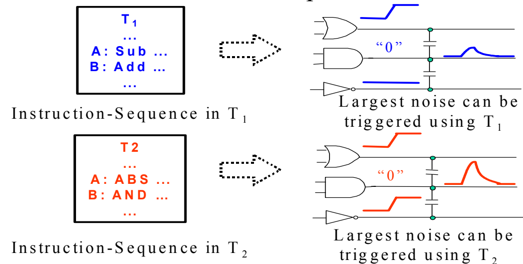


Figure 4: Instruction sequence affects test vector quality

In addition, the instruction sequence in a test program template predefines the best vectors that can be delivered

to detect AC failures. An example is shown in Figure 4. For the same victim wire in the same MUT, test generations were performed with different constraints from template T_1 and T_2, respectively. Here T_2 has the potential to trigger a larger noise than T_1. In the test generation process for stuck-at faults, a fault can be removed from fault list if a test vector is successfully generated under any template. On the contrary, for crosstalk fault we have to enumerate all the possible instruction sequences for every fault, keeping records of test results and compare these results to find the instruction sequence causing the largest noise. The enumeration-based method becomes extremely expensive.

3. NON-ENUMERATION TEST GENERATION METHODOLOGY

Facing these challenges, a non-enumeration methodology has been developed to simplify the searching process and automate the test program generation. The proposed methodology eliminates the enumeration of test program templates and further reduces the complexity of constrained test generation.

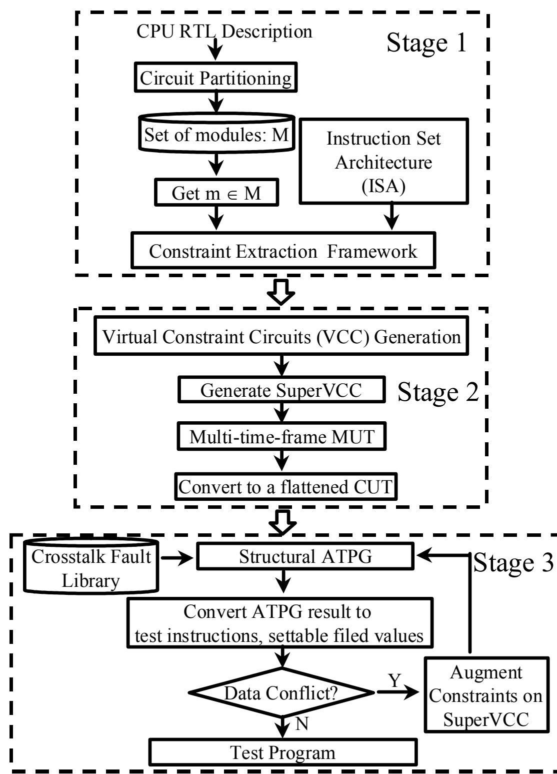


Figure 5: Test program generation flow

As shown in Figure 5, the proposed method has three stages. Stage 1 performs module partitioning and constraint extraction for all instructions. A processor is partitioned into a set of modules based on its RTL description. Then a simulation-based method is used to extract controllability and observability constraints for each MUT. The constraints are extracted by deriving input/output mapping functions using regression analysis techniques, as described in [15].

Stage 2 prepares a CUT to constrain the module-level ATPG process, containing the MUT sandwiched between the controllability and observability VCCs. There are two instruction slots (A and B) in the instruction sequence to be filled with appropriate instructions. Since instructions from different timeframes have different VCCs, the ATPG tool needs to handle a changing CUT across the timeframes. Although a time-varying CUT is required to represent constraints imposed by a sequence of instructions, the "right" instruction sequence for triggering the largest noise effect is not known *a-priori*. In [15], the ATPG tool considers single-timeframe constraints by enumerating all CUTs with VCCs representing constraints imposed by individual instructions in the ISA. The same enumeration method is not feasible for multi-timeframe test generations, as the number of CUTs representing all possible instruction combinations grows exponentially as the length of the sequence grows.

To overcome this difficulty, we propose the concept of ***Super Virtual Constraint Circuit*** (SuperVCC). Since ATPG tools are good at making decisions based on structural information, we shift the responsibility of selecting the instruction sequence to the ATPG tool by making the choices of all instructions available to the ATPG tool in the form of a SuperVCC. A SuperVCC encapsulates all VCCs, representing constraints imposed by the entire ISA in a single timeframe. As shown in Figure 6, a multiplexer and a demultiplexer are introduced to combine individual VCCs. A "select" signal is added to control the MUX and DEMUX. Thus the ATPG tool can later choose an instruction implicitly by assigning values to the "select" signal. After the generation of the SuperVCC, the MUT is duplicated to enable test generation in multiple timeframes. A SuperVCC is attached to the MUT in each timeframe, representing constraints imposed by instructions from the corresponding timeframe.

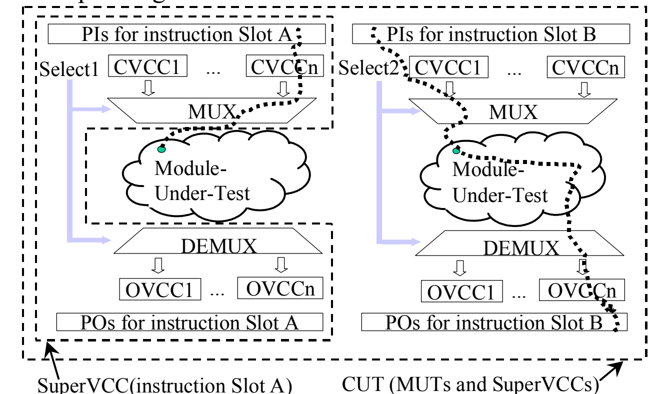


Figure 6: CUT – MUTs augmented with SuperVCCs

The circuit consisting of the MUT and the SuperVCC

spanning over multiple timeframes, forms the new CUT seen by an ATPG tool to generate tests for crosstalk faults [18]. Thus, instead of handling N^2 CUTs (2 timeframes to consider for crosstalk faults), where N is the number of instructions in the ISA, the ATPG tool now only need to generate test for a single CUT. Note that the CUT itself does not imply any particular instruction sequence. Rather, the selection of the instruction sequence triggering the desired noise effect is done by ATPG in Stage 3, without enumerating different CUTs.

SuperVCC is a virtual circuit whose only objective is to present constraints to the ATPG tool. Simply connecting all VCCs together with MUX will make the SuperVCC too complex and difficult for ATPG tool to handle. Fortunately, common signals and logic redundancies can be exploited to further reduce the complexity of the SuperVCC. Common settable fields and observable fields of VCCs are merged to reduce the total number of PIs and POs. Many VCCs have signals either attach to logic "1" or logic "0", hence the MUX and DEMUX can be tremendously trimmed. For instructions belonging to one category (e.g. add, sub), frequently their VCCs share common logics and signals. Thus, logic synthesis tool is used to further simplify the SuperVCC. Figure 7 shows these mechanisms to reduce the logic complexity of SuperVCC for input constraints.

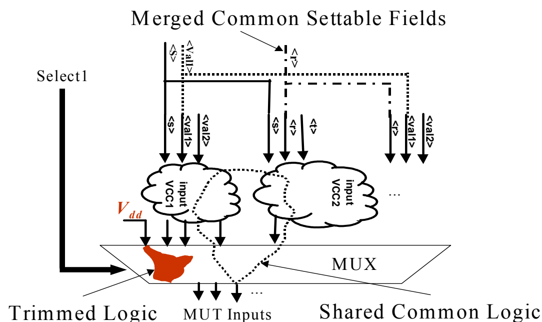
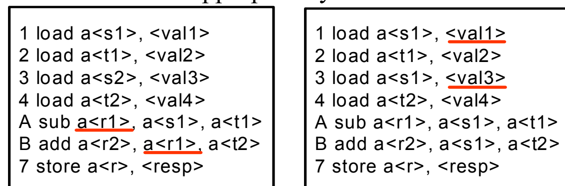


Figure 7: Logic reduction for SuperVCC

In Stage 3, constrained ATPG is performed. The ATPG outputs specify values for the "select" signals and settable fields of the "selected" instructions. The test vectors justify the logic values for the victim, trigger aggressors and sensitize a propagation path (shown as dotted line in Figure 6). The ATPG results are then converted to a test instruction sequences by mapping the values assigned to the "select" signals to corresponding instructions. Peripheral instructions are automatically inserted to setup values for settable fields and save the test results.

Besides the complexity challenges discussed above, there are data-dependency related issues. Figure 8 shows two typical situations. In Figure 8 (a), test instruction B needs data from register r1, which is being calculated by instruction A. However, in pipelined microprocessors, this data is not available unless pipeline forwarding is enabled. Figure 8 (b) shows for register s1, instruction A and instruction B require different values and one of the tests cannot be delivered appropriately.



(a) Pipeline Forwarding (b) Data Conflict

Figure 8: Conflicts induced by data correlation

To solve these problems, conflict-aware test program generation processes were developed. One way to ensure the data-dependency consistency is to implement the correlation check in the SuperVCC. Another way is to use a simple SuperVCC without data correlation check and utilize a conflict-aware value assignment process to intelligently make use of *don't care* signals in the ATPG results. Data correlation consistency is ensured by careful assignments on *don't care* signals, leading to the access of non-conflict sources and destinations in the test instructions. If data conflict could not be resolved by the intelligent value assignments, the SuperVCC will be incrementally augmented with (only) the necessary constraints, forcing ATPG tool to avoid conflicts already occurred.

4. EXPERIMENTAL RESULTS

We applied the proposed methodology to the Xtensa processor from Tensilica Inc. [16]. The Xtensa processor is a commercial configurable and extensible RISC processor with 5 pipeline stages and 81 core instructions. A typical configuration of Xtensa processor was synthesized using Leonardo Spectrum™ [19].

Experiments were conducted on a large module, called EX1, which is a logic cone leading to the result of the execution stage in the pipeline. This logic cone has one 32-bit output port and 81 input ports corresponding to 335 bits. EX1 contains basic building blocks for completing arithmetic/relational/logical operations. It also contains logics for pipeline forwarding. A layout of EX1 was generated in IC Station® [20], and parasitic parameters were extracted by xCalibre® [21]. A total of 998 potential crosstalk-fault candidates were identified using static noise analysis [22].

We first performed simulation-based constraint extractions and generated individual VCCs. A SuperVCC was automatically generated by a *PERL* script and optimized by Leonardo Spectrum™. Table 1 shows the advantage of the SuperVCC method as compared with the enumeration-based method [15], in terms of number of CUTs that the ATPG tool needs to handle and test generation time. For the enumeration-based method, 88,209 CUTs has to be constructed. For each fault, ATPG needs to be performed on *all* CUTs repeatedly and the ATPG results have to be compared to select the best

instruction sequence. As shown in Table 1, the SuperVCC-based methodology tremendously accelerates the test generation process. Considering the extra processing time that would have been needed for constructing these 88,209 CUTs and the ATPG runs that have to be performed on all CUTs for each fault, the gain in efficiency is magnificent.

Table 1: Complexity of SuperVCC and enumeration

Methods	# of CUTs	ATPG Time (sec.)
SuperVCC	1	1376
Enumeration	88,209	3.7E10 (estimated)

Table 2 shows that the complexity reduction techniques such as logic optimization and merging common IOs can tremendously reduce the complexity of the generated SuperVCC. We compared gate count (2-input NAND gate), number of internal signals and PIs.

Table 2: Effectiveness of complexity reduction techniques

SuperVCC Generation Methods	Gate Count	Internal Signals	# of PI
With reduction	1,144	1,358	540
Without reduction	34,560	14,128	13,244

Test quality is also examined. Constrained test generations were performed for the 988 faults. The test program contains 6914 instructions. For comparison, structural tests were also generated without constraints (assuming enhanced scan), using the same ATPG engine. To evaluate the efficiency of the proposed method, the number of triggered aggressors, total ATPG run time, and number of aborted faults are listed in Table 3. The effectiveness of test vectors was measured by the total weight (normalized noise magnitude) of the triggered aggressors [18]. As shown in column one of Table 3, the average weight triggered by SBST is 91% of the average weight triggered by enhanced scan. Note that, even though enhanced scan can trigger weights slightly higher than SBST, many of the larger weights can only be triggered in non-functional mode. Hence, enhanced scan provides a loose upper bound. Having to consider instruction-imposed constraints, the SBST method requires an ATPG time less than triple of the time required by scan test generation.

Table 3: ATPG results with and without ISA constraints

Methods	Avg. triggered weights	ATPG time (sec.)	# of aborted faults
SBST	18.489	1376	186
Scan	20.14	425	46

5. CONCLUSIONS

In this work, we developed an efficient generic test program generation methodology that enables software-based self-test for crosstalk in processors. The proposed methodology dramatically reduces the complexity for multiple-timeframe test generation in SBST. It utilizes the novel concept of SuperVCC to combine structural ATPG with instruction-level constraints. Guided by SuperVCC and physical defects, the selection of test instructions and settable field values are accomplished by structural ATPG. Though illustrated on crosstalk faults, the methodology can also be applied to other dynamic fault models that require the at-speed application of test vector sequences.

6. REFERENCES

[1] B. I. Dervisoglu and G. E. Stong, "Design for testability: using scan path techniques for path-delay test and measurement", *Proc. of Int. Test Conf.*, 1991, pp. 365-374.

[2] Y. Zorian, "A Distributed BIST Control Scheme for Complex VLSI Devices," *Proc. of VLSI Testing Symposium*, pp. 4-9, 1993.

[3] S. Wang, "Generation of Low Power Dissipation and High Fault Coverage Patterns for Scan-Based BIST," *International Test Conference*, 2002, pp. 834-843.

[4] T. M. Mak, "Limitations of Structural Delay Tests", Gigascale Systems Research Center (GSRC) talk topic, 2003.

[5] International Technology Roadmap for Semiconductors (ITRS), 2001Edition.

[6] J. Shen and J. A. Abraham, "Native model functional test generation for processors with applications to self test and design validation," *Proc. Intl. Test Conf.*, Oct. 1998, pp.990-999

[7] K. Batcher and C. Papchristou, "Instruction randomization self test for processor cores," *Proc. 17th IEEE VLSI Test Symp.*, April 1999, pp.34-40.

[8] P. Parvathala, K. Maneparambil and W. Lindsay, "FRITS – A microprocessor functional BIST method," *Proc. Intl Test* Conf., Oct 2002, pp.590-598.

[9] L. Chen and S. Dey, "DEFUSE: A Deterministic Functional Self-Test Methodology for Processors," *Proc. 18th IEEE VLSI Test Symp.*, May 2000, pp.255-262.

[10] L. Chen and S. Dey, "Software-Based self-testing methodology for processor cores," IEEE Trans. Computer-Aided Design, vol.20, no.3, March 2001, pp.369-380.

[11] P. Vishahantaiah, J. Abraham and M. Abadir, "Automatic test knowledge extraction from VHDL (ATKET)," *Proc. 29th Design Automation Conference*, June 1992, pp.273-278.

[12] R. Tupuri, A. Krishnamachary and J. Abraham, "Test Generation for Gigahertz Processors Using an Automatic Functional Constraint Extractor," *Proc. 36th Design Automation Conference*, June 1999, pp.647-653.

[13] W.-C. Lai, *Embedded Software-Based Self-Test for System-on-a-Chip Design*, Ph.D. thesis, UC Santa Barbara, March 2002.

[14] W.-C. Lai, A. Krstic and K.-T. Cheng, "Test program synthesis for path delay faults in microprocessor cores," *Proc. Intl. Test Conf.,* Oct. 2000, pp.1080-1089

[15] L. Chen, S. Ravi, A. Raghunathan and S. Dey, "A Scalable Software-Based Self-Test Methodology for Programmable Processors," *Proc. Design Automation Conference*, June 2003, pp.548-553.

[16] *XtensaTM Microprocessor Overview Handbook*, Tensilica Inc, Aug. 2001.

[17] R. Tupuri and J. Abraham, "A Novel Functional Test Generation Method for Processors using Commerical ATPG," *Proc. Intl. Test Conference*, Nov. 1997, pp.743-752

[18] X. Bai, S. Dey and A. Krstic, "ATPG for Crosstalk using Hybrid Structural SAT", *proc. Intl. Test Conference*, Oct. 2003, pp. 112-121.

[19] LeonardoSpectrumTM, Mentor Graphics Corp.

[20] IC Station®, Mentor Graphics Corp.

[21] xCalibre®, Mentor Graphics Corp.

[22] X. Bai, R. Chandra, S. Dey and P.V. Srinivas, "Noise-Aware Driver Modeling," *Proc.* of *International Symposium on Quality Electronic Design*, March 2003, pp.177-182.