Low-level logic fault testing ASIC simulation environment

by

Mansour H. Assaf, Leslie-Ann Moore, Sunil R. Das, Satyendra N. Biswas and Scott Morton

reprinted from

WORLD JOURNAL OF ENGINEERING

VOLUME 11 NUMBER 3 2014

MULTI-SCIENCE PUBLISHING COMPANY LTD.

World Journal of Engineering 11(3) (2014) 279-286

World Journal of
Engineering

Low-level logic fault testing ASIC simulation environment

Mansour H. Assaf[1], Leslie-Ann Moore[2], Sunil R. Das[3,4,*], Satyendra N. Biswas[5] and Scott Morton[3]

[1]School of Engineering and Physics, University of the South Pacific, Suva 19128, Fiji
[2]Center for Information and Communication Technology, University of Trinidad and Tobago, Arima, Trinidad, West Indies
[3]Department of Computer Science, College of Arts and Sciences, Troy University, Montgomery, AL 36103, USA
[4]School of Information Technology and Engineering, Faculty of Engineering, University of Ottawa, Ottawa, ON K1N 6N5, Canada
[5]School of Engineering and Technology, Kaziranga University, Jorhat, Assam 785006, India
**E-mail: sdas@troy.edu*

(Received July 21, 2013; accepted March 23, 2014)

Abstract

A low-level logic fault test simulation environment targeted towards application-specific integrated circuits (ASICs) in particular is proposed in this paper. The simulation environment emulates a typical built-in self-testing (BIST) environment with test pattern generator (TPG) that sends its outputs to a circuit (core) under test (CUT) and the output streams from the CUT are fed into an output response analyzer (ORA). The developed simulator is very suitable for testing embedded digital intellectual property (IP) cores-based systems. The paper describes the total test architecture environment, including the application of the logic fault simulator. Results on simulation on some specific International Symposium on Circuits and Systems (ISCAS) 85 combinational and ISCAS 89 sequential benchmark circuits are provided as well for appraisal.

Key words: *Application-specific integrated circuit, built-in self-testing, circuit under test, intellectual property cores, low-level logic fault test simulation, test pattern generator*

1. Introduction

With increasing complexity in systems design concurrent with enhanced levels of integration densities, better and more efficient methods of testing to ensure reliable operations of chips, mainstay of today's many sophisticated digital systems, are evidently a necessity (Assaf, 2003; Bardell *et al*., 1987; Bellows and Hutchings, 1998; Chakrabarty, 1995, 2005; Das *et al*., 1998, 2001, 2001; Gherman *et al*., 2006; Guccione and Levy, 1998; Huang *et al*., 2001; Jone and Das, 1991; Karpovsky and Nagvajara, 1990; Levi and Guccione, 1999; Li and Robinson, 1987; McCluskey, 1985; Pouya and Touba, 1998; Pradhan and Gupta, 1991; Rajsuman, 2000; Reddy *et al*., 1988; Saluja and Karpovsky, 1983; Savir, 1996; Sundararajan and Guccione, 2000). The very concept of testing has a broad applicability, and finding the most effective

ISSN: 1708-5284

testing techniques that can guarantee correct system performance is of immense practical relevance. Generally, the price of testing integrated circuits (ICs) is rather prohibitive, accounting for 35–55% of their total manufacturing cost. Furthermore, testing a chip is also time-consuming, taking up to about one-half of the total design cycle time. The amount of time available for manufacturing, testing and marketing a product, on the other hand, is constantly on the decline. Also, as a result of diminishing trade barriers and global competition, customers now demand products of better quality at lower cost. In order to achieve this higher quality at reduced cost, obviously the testing methods have to be improved. The conventional testing techniques of digital circuits require application of test patterns generated by a test pattern generator (TPG) to the circuit (core) under test (CUT) and comparing the response with known correct response (Das *et al*., 1998, 2001, 2001). For large circuits, because of higher memory requirements for the fault-free responses, the customary test procedures hence become rather expensive, and so alternative approaches are sought to reduce the amount of needed storage (Das *et al*., 2001). Built-in self-testing (BIST) is a design methodology that provides the capability of solving many of the problems otherwise encountered in testing digital systems. It combines the concepts of both built-in test (BIT) and self-test (ST) into one, termed BIST. In BIST, test generation, test application and response verification are all accomplished through built-in hardware, which allows different parts of a chip to be tested in parallel, thereby reducing the required testing time, besides eliminating the necessity for external test equipment. As the cost of testing is becoming the single major component of the manufacturing expense of a new product, BIST thus tends to reduce the manufacturing and maintenance costs through improved diagnosis. Several companies such as Motorola, AT&T, IBM and Intel have incorporated BIST in many of their products (Das *et al*., 2001). AT&T, for example, has incorporated BIST into more than 200 of their IC chips. The three large programmable logic arrays (PLAs) and microcode read-only memory (ROM) in the Intel 80386 microprocessor were all built-in self-tested (Das *et al*., 2001). The general purpose microprocessor chip, Alpha AXP21164 and Motorola microprocessor 68020, were also tested using BIST techniques (Das *et al*., 2001). More recently, Intel, for its Pentium Pro architecture microprocessor, with its unique requirements of meeting very high production goals, superior performance standards and impeccable test quality put strong emphasis on its design-for-test (DFT) direction (Das *et al*., 2001). A set of constraints, however, limits Intel's ability to tenaciously explore DFT and test generation techniques, *viz*. full-scan or partial-scan or scan-based BIST (Das *et al*., 2001). AMD's K6 processor is a reduced instruction set computer (RISC) core named enhanced reduced instruction set computer RISC86 microarchitecture (Das *et al*., 2001). K6 processor incorporates BIST into its DFT process. Each random-access memory (RAM) array of K6 processor has its own BIST controller. BIST executes simultaneously on all of the arrays for a predefined number of clock cycles that ensures completion for the largest array. Hence, BIST execution time depends on the size of the largest array (Das *et al*., 2001). AMD uses commercial automatic test pattern generation tool to create scan test patterns for stuck-faults in their processor. The DFT framework for 500-MHz IBM S/390 microprocessor utilizes a wide range of tests and techniques to guarantee superb reliability of components within a system. Register arrays are tested through the scan chain, whereas nonregister memories are tested with programmable RAM BIST. Hewlett-Packard's PA8500 is a 0.25 μm superscalar processor that achieves fast but thorough test with its cache test hardware's ability to perform March tests, which is an effective way to detect several kinds of functional faults. Digital's Alpha 21164 processor combines both structured and *ad hoc* DFT solutions, for which a combination of hardware and software BIST was adopted. Sun Microsystems' UltraSparc processor incorporates several DFT constructs as well. The achievement of its quality performance coupled with reduced chip area conflicts with a design requirement that is easy to debug, test and manufacture (Das *et al*., 2001).

BIST is widely used to test embedded regular structures that exhibit a high degree of periodicity such as memory arrays, *viz*. static RAMs (SRAMs), ROMs, first-in first-outs (FIFOs) and registers. This type of circuits does not require complex extra hardware for test generation and response compaction. Also, including BIST in these circuits can guarantee high fault coverage (FC) with zero-aliasing (Chakrabarty, 1995; Das *et al*., 2001). Unlike regular circuits, random-logic circuits cannot be adequately tested only with BIST techniques, since generating adequate on-chip test sets using simple hardware is a difficult task to be accomplished. Besides, since test responses generated by random-logic circuits seldom exhibit regularity, it is extremely difficult to ensure zero-aliasing compaction. Therefore, random-logic circuits are usually tested using a combination of BIST, scan design techniques and external test equipment.

A typical BIST environment, as shown in Figure 1, uses a TPG that sends its outputs to a CUT and output streams from the CUT are fed into an output response analyzer (ORA). A fault is detected if the test sequence is different from the response of the fault-free circuit. The test data analyzer is composed of a response compaction unit, storage for the fault-free responses of the CUT and comparator. In order to reduce the amount of data represented by the fault-free and faulty CUT responses, data compression is used to create signatures (short binary sequences) from the CUT and its corresponding fault-free circuit. Signatures are compared and faults are detected if a match does not occur.

BIST techniques may be used during normal functional operating conditions of the unit under test (on-line testing), as well as when a system is not carrying out its normal functions (off-line testing). In the case where detecting real time errors is not that important, systems, boards and chips can be tested in off-line BIST mode. BIST techniques use pseudoexhaustive or pseudorandom test patterns, or sometimes on-chip storing of reduced or compact test sets. Today, testing logic circuits exhaustively is seldom used, since only a few test vectors are needed to ensure full FC for single stuck-line faults (Das *et al.*, 2001, 2001). Reduced pattern test sets can be generated using existing algorithms such as FAN and others (Chakrabarty, 1995). BIT generators can often generate such reduced test sets at low cost, making BIST techniques suitable for on-chip self-testing.

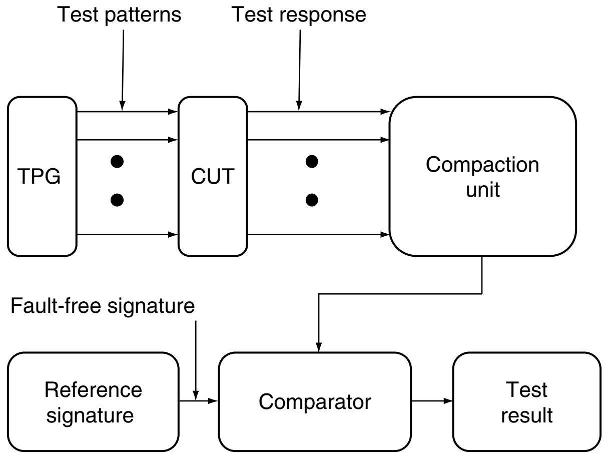

Fig. 1. A typical BIST environment.

In the subject paper, a low-level logic fault simulation environment for built-in self-testing of digital IP cores-based systems is proposed.

2. Testing digital embedded cores-based systems

The testing of ASICs and IP cores poses a serious challenge with respect to the overall test cost. This is particularly so due to the enormous difficulty of test access to the embedded cores, test application time and high test data volume (Chakrabarty, 2005). Also, testing at speed becomes a problem for the automatic test equipment in order to keep up with the system clock rate. A wrapper is used to isolate the IP cores from the environment, while a test access mechanism (TAM) is required for accessing and mapping the input vector patterns into the IP cores or system-on-chips (SOCs) (Chakrabarty, 1995; Huang *et al.*, 2001). Figure 2 illustrates the basic SOC structure with its corresponding testing environment.

3. Test architecture environment

An outline of the actual design and realization of the test environment is discussed now. The testing system tends to incorporate built-in test technology into a chip to guarantee high testability. The test hardware can be a collection of test circuits, power supplies, measuring outfits and transition devices that have digital-to-analog (D/A) and analog-to-

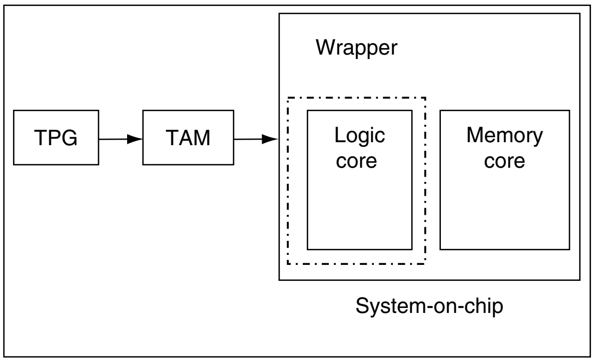

Fig. 2. An SOC structure.

digital (A/D) functions. A buffer memory is used to store test patterns and output response. Generally, these circuits can be described by such hardware description languages (HDLs) as very high speed integrated circuit (VHSIC) hardware description language (VHDL), Verilog HDL (Xilinx, 1999), netlist or others. Here, the International Symposium on Circuits and Systems or ISCAS benchmark circuits used are described in their netlist formats.

The objective of the test software here is to implement the automation of the computerized testing process. The programs embedded onto the chip can take control of the function of testing and apply test patterns to specific ports. Also, it can compare the produced output response with the stored fault-free responses. The comparison results are recorded for further analysis. Frankly, the test software will undertake to carry out the test automation and test simulation. In short, the primary aim in this context is to realize the fulfilment of the logic circuit test simulation environment, to carry out the verification scheme and to compare the results to desired values to make certain that the test mechanism is working as expected.

The test environment is designed to handle both ASICs and chips consisting of combinations of ISCAS 85 combinational and ISCAS 89 sequential benchmark circuits. It utilizes a pseudorandom input test pattern generator in combination with pseudoexhaustive input test vectors. The fault simulation programs FSIM (Lee and Ha, 1993), ATALANTA (Lee and Ha, 1991) and HOPE (Lee and Ha, 1992) were used here to generate the required input test vectors, both pseudorandomly and deterministically. However, COMPACTEST program (Pomeranz *et al*., 1991) was not used in the current study, though it was used by the authors in many of their prior research works. Figure 3 introduces a system level implementation of the architecture of such a digital test environment.

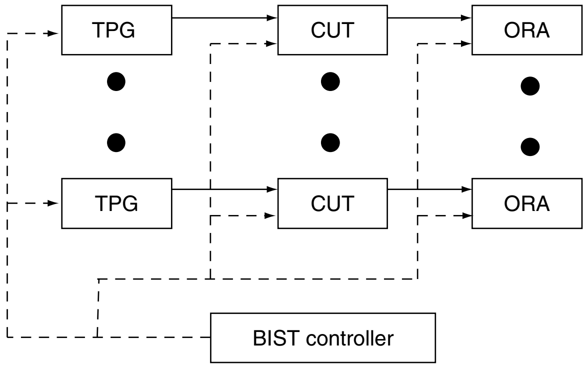

Fig. 3. Test environment.

The BIST controller selects the CUT for testing and also monitors test scheduling in the test mode. Each core under test has its own set of input test vectors which feed the cores, while the produced response is verified by the output response analyzer.

Figure 4 provides a diagrammatic representation of the corresponding system test architecture. The controller is needed for test scheduling and core selection. The role of the test access mechanism is to help input test application and output response observation. The wrapper is used to isolate the core from its environment during testing. The following is a snapshot of the test process.

$$\text{If } x = 0 \text{ and } y = 0 \rightarrow \text{normal operation mode;}$$

$$\text{If } x = 1 \text{ and } y = 1 \rightarrow \text{test mode;}$$

$$X = (x_1, x_2, x_3, \ldots, x_m);\ Y = (y_1, y_2, y_3, \ldots, y_n);$$

$$n \neq m.$$

The modular isolation of the various cores on the SOC is carried out first; the output response from a core under test is next fed to the response analyzer for fault coverage evaluation.

4. Results on implementation

In order to study the feasibility of the proposed logic fault test simulation environment, experiments were conducted on various ISCAS 85 combinational and ISCAS 89 sequential benchmark circuits. As shown in Table 1, 38 circuits from ISCAS 85 and ISCAS 89 families were selected for simulation as digital IP cores to implement the SOCs. The simulation results confirm a number of interesting observations regarding percentage fault coverage, memory usage, simulation CPU time needed to test the core under test and CPU time taken for testing the SOCs. The IP cores were first isolated and the total isolation simulation time was computed. The data include results from individual cores that were merged to form a modular partition while simulated. Figure 5 provides a flow chart representation of fault injection and fault coverage under the proposed scheme.

Figure 6 shows fault coverage of 11 isolated combinational benchmark circuits tested pseudorandomly, the lowest coverage being 81.616%, bringing the fault coverage average to

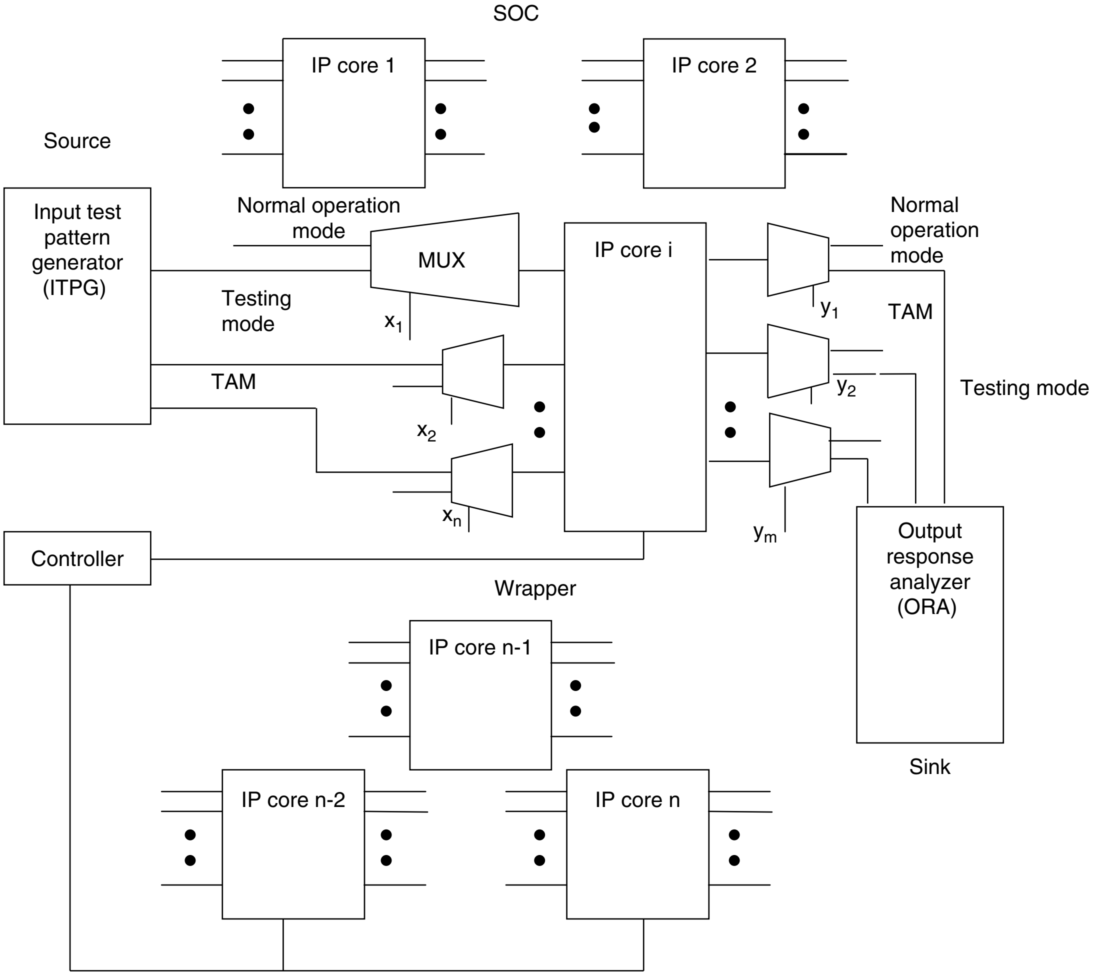

Fig. 4. A diagrammatic representation of the system test architecture.

Table 1.
ISCAS 85 and ISCAS 89 benchmark circuits

Benchmark Circuits	
c17.bench	s526.bench
c432.bench	s635.bench
c499.bench	s641.bench
c880.bench	s713.bench
c1355.bench	s820.bench
c1908.bench	s832.bench
c2670.bench	s938.bench
c3540.bench	s953.bench
c5315.bench	s967.bench
c6288.bench	s991.bench
c7552.bench	s1196.bench
s27.bench	s1238.bench
s298.bench	s1269.bench
s344.bench	s1423.bench
s349.bench	s1488.bench
s382.bench	s1494.bench
s386.bench	s1512.bench
s444.bench	s3271.bench
s510.bench	s3330.bench

92.895%. The sequential portions, however, have fault coverage that is less than half of the combinational average, with the lowest circuit coverage being at 39.739%. Figure 7 furnishes the comparative memory usage for combinational and sequential benchmark circuits. The maximum storage space needed is that for the core c7552 which is 169267 kilobytes, while for the core s3271, it is 168611 kilobytes. The number of faults detected compared to the total number of faults injected for various benchmark circuits is portrayed in Figure 8.

The fault coverage was measured in both the pseudorandom and pseudoexhaustive testing modes. The results demonstrate that the fault coverage is higher in the pseudorandom mode as contrasted to the pseudoexhaustive mode. The ratio of the fault coverage is projected to be at an average of 92.695/92.895 for pseudoexhaustive and pseudorandom testing modes, respectively.

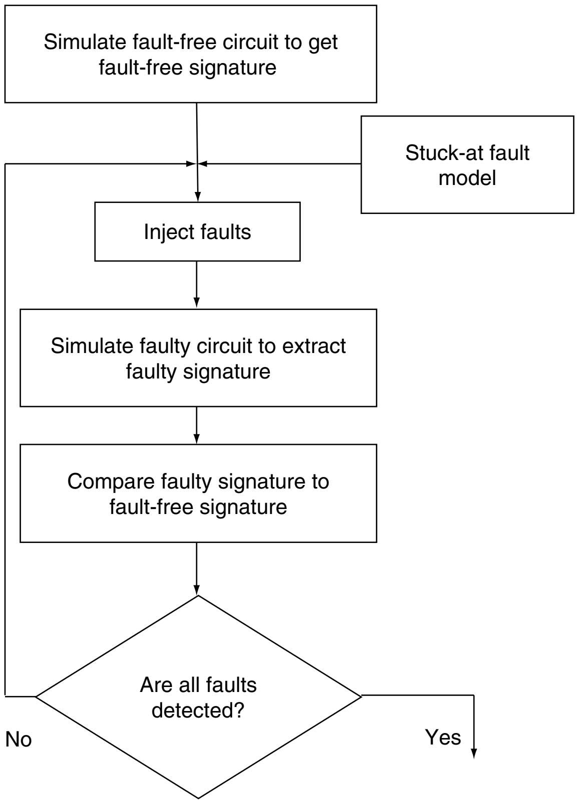
Simulate fault-free circuit to get fault-free signature
Stuck-at fault model
Inject faults
Simulate faulty circuit to extract faulty signature
Compare faulty signature to fault-free signature
Are all faults detected?
No
Yes

Fig. 5. Fault injection and coverage – a flow chart.

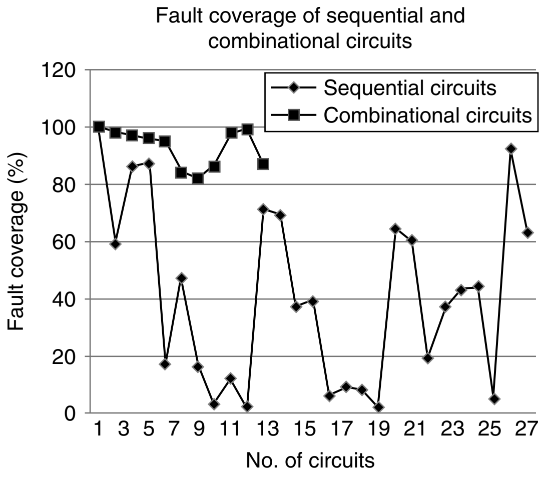

Fig. 6. Percentage fault coverage for sequential and combinational benchmark circuits.

The CPU time taken to simulate the different cores in pseudorandom and pseudoexhaustive testing modes is provided in Figure 9. The simulation CPU time for pseudorandom testing is much higher as compared to that of the pseudoexhaustive testing.

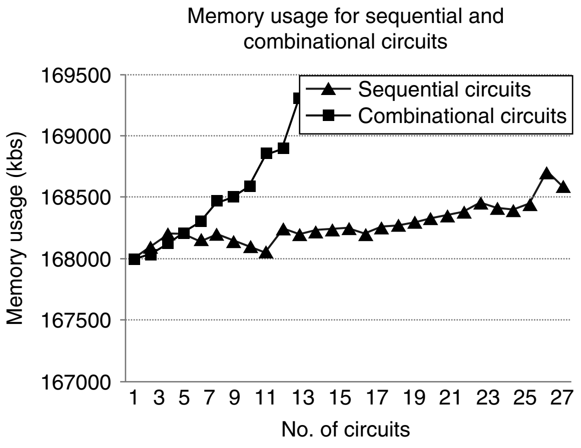

Fig. 7. Memory usage for sequential and combinational benchmark circuits.

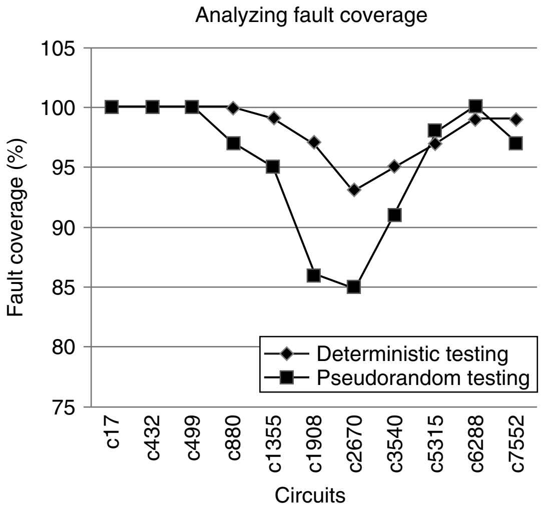

Fig. 8. Percentage fault coverage in deterministic and pseudorandom testing modes.

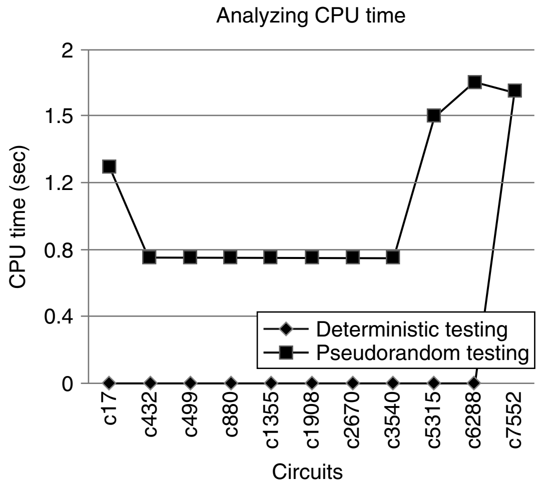

Fig. 9. Simulation CPU time for deterministic and pseudorandom testing.

5. Concluding remarks

To conclude, evidently further studies need to be undertaken to develop a comprehensive fault test simulation environment for digital IP cores with the potential to handling faults other than single stuck-line faults. The proposed scheme seems to advance a powerful tool to gather statistical data for logic circuits as well. This low-level simulation environment can be extended to handle logic circuits described in other hardware description languages, *viz*. VHDL, Verilog HDL (Xilinx, 1999), etc. In recent SOC designs, many independent modules can be contained in a single chip in IP cores-based fashion. These IP cores can be central processing units (CPUs), memories, digital signal processors (DSPs) and different kinds of communication modules. One of the advantages of IP cores is that they speed up the design cycle of large complex system chips, achieving a shorter time-to-market. But it raises good challenges for testing also.

Finally, there are many ways to ensure that a design is functioning correctly in hardware, but one of the most efficient and reliable methods is centered upon design-for-testability approach. The current research basically augments that philosophy to include digital ASIC testing in differing environments.

Acknowledgments

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under Grant A 4750.

References

Assaf M.H., 2003. Digital Core Output Test Data Compression Architecture Based on Switching Theory Concepts. Ph.D. Dissertation, School of Information Technology and Engineering, University of Ottawa, Ottawa, ON, Canada.

Bardell P.H., McAnney W.H., Savir J., 1987. Built-In Test for VLSI: Pseudorandom Techniques, Wiley Interscience, New York, U.S.A.

Bellows P., Hutchings B., 1998. JHDL–An HDL for reconfigurable systems. Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines, 175–184.

Chakrabarty K., 1995. Test Response Compaction for Built-In Self-Testing. Ph.D. Dissertation, Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI, U.S.A.

Chakrabarty K., 2005. Low-cost modular testing and test resource partitioning for SOCs. *IEE Proceedings for Computers and Digital Techniques* **152**, 427–441.

Das S.R., Petriu E.M., Barakat T., Assaf M.H., Nayak A.R., 1998. Space compaction under generalized mergeability. *IEEE Transactions on Instrumentation and Measurement* **47**, 1283–1293.

Das S.R., Assaf M.H., Petriu E.M., Jone W.B., Chakrabarty K., 2001. A novel approach to designing aliasing-free space compactors based on switching theory formulation. Proceedings of the IEEE Instrumentation and Measurement Technology Conference **1**, 198–201.

Das S.R., Ramamoorthy C.V., Assaf M.H., Petriu E.M., Jone W.B., 2001. Fault tolerance in systems design in VLSI using data compression under constraints of failure probabilities. *IEEE Transactions on Instrumentation and Measurement* **50**, 1725–1747.

Gherman V., Wunderlich H.J., Schloeffel J., Garbers M., 2006. Deterministic logic BIST for transition fault testing. Proceedings of the IEEE European Test Symposium 123–130.

Guccione S.A., Levi D., 1998. XBI: A Java-based interface to FPGA architecture. Proceedings of the SPIE **3526**, 97–102.

Huang J.R., Iyer M., Cheng K.T., 2001. A self-test methodology for IP cores in bus-based programmable SOCs. Proceedings of the IEEE VLSI Test Symposium 198–203.

Jone W.B. and Das S.R., 1991. Space compression method for built-in self-testing of VLSI circuits. *International Journal of Computer Aided VLSI Design* **3**, 309–322.

Karpovsky M., Nagvajara P., 1990. Optimal robust compression of test responses. *IEEE Transactions on Computers* **39**, 138–141.

Lee H.K., Ha D.S., 1991. An efficient forward fault simulation algorithm based on the parallel pattern single fault propagation. Proceedings of the IEEE International Test Conference, 946–955.

Lee H.K., Ha D.S., 1992. HOPE: An efficient parallel fault simulator for synchronous sequential circuits. Proceedings of the IEEE/ACM Design Automation Conference, 336–340.

Lee H.K., Ha D.S., 1993. On the generation of test patterns for combinational circuits. Technical Report 12–93, Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, VA, U.S.A.

Levi D., Guccione S.A., 1999. Genetic FPGA: A Java-based tool for evolving stable circuits. Proceedings of the SPIE **3844**, 114–121.

Li Y.K., Robinson J.P., 1987. Space compression method with output data modification. *IEEE Transactions on Computer-Aided Design* **6**, 290–294.

McCluskey E.J., 1985. Built-in self-test techniques. *IEEE Design & Test of Computers* **2**, 21–28.

Pomeranz I., Reddy L.N., Reddy S.M., 1991. COMPACTEST: A method to generate compact test sets for combinational circuits. Proceedings of the IEEE International Test Conference 194–203.

Pouya B., Touba N.A., 1998. Synthesis of zero-aliasing elementary-tree space compactors. Proceedings of the IEEE VLSI Test Symposium 70–77.

Pradhan D.K., Gupta S.K., 1991. A new framework for designing and analyzing BIST techniques and zero aliasing compression. *IEEE Transactions on Computers* **40**, 743–763.

Rajsuman R., 2000. System-on-a-Chip: Design and Test, Artech House, Boston, MA, U.S.A.

Reddy S.M., Saluja K.K., Karpovsky M.G., 1988. Data compression technique for test responses. *IEEE Transactions on Computers* **37**, 1151–1156.

Saluja K.K., Karpovsky M., 1983. Testing computer hardware through compression in space and time. Proceedings of the IEEE International Test Conference 83–88.

Savir J., 1996. Reducing the MISR size. *IEEE Transactions on Computers* **45**, 930–938.

Sundararajan P., Guccione S.A., 2000. XVPI: A portable hardware/software interface for virtex. Proceedings of the SPIE **4212**, 90–95.

The JBits SDK, Xilinx, Inc., http://www.xilinx.com/products/jbits/

Xilinx, Inc., 1999. The Programmable Logic Data Book, San Jose, CA, U.S.A.