An Interactive Approach for Trading Test Time, Area, and Fault Coverage in Testable Synthesis

Soha Baz and Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010, Lebanon

Keywords: Electronics, High-Level Synthesis, Testing, Fault Simulation

Abstract

This paper presents a method for interactive test synthesis for RTL designs. The method provides a mechanism to redesign RTL datapaths into testable ones by exploring possible tradeoffs among area, test time and test quality. Designs are first mapped into *structurally* testable templates. Designers can then interact with the system through a graphical user interface in order to interactively modify and select test points. The system diagnoses and pin-points fault simulation weaknesses in the design.

1. INTRODUCTION

VLSI chips designers have long realized the importance and advantages of incorporating testability early in the design process. Synthesis is defined as the translation of a behavioral description into a structural one. The main difficulty in this process is that most behaviors have no implied architecture. Thus it is rather impossible to develop synthesis algorithms that will generate the same quality from all possible design descriptions. One of the main tasks of high-level synthesis is to find the structure that best meets the constraints while minimizing other costs.

High-level synthesis has several advantages. First, it increase designers productivity by exploring the design space in a relatively short time. The resulting synthesized designs are correct by construction. Another advantage is that it reduces silicon literacy requirement since designers do not have to be familiar with the details and peculiarities of technology specific aspects.

Built-In Self Test (BIST) is a "test per clock" DFT technique that allows a circuit to test itself by embedding external tester features such as test pattern generation and output response analysis, into the chip itself. Even though BIST can reduce the test application time in addition to help in delay testing and defect coverage, area overhead and performance degradation in addition to increased design time are often cited as reasons for the limited use of BIST. During self-test,

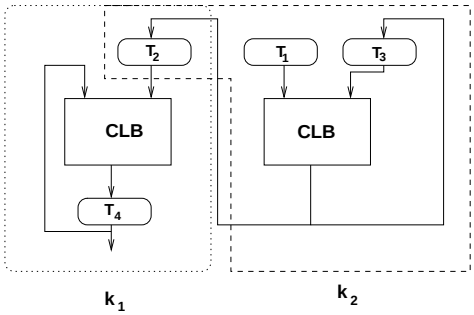

Figure 1. (a) Testable Functional Block, (b) Self Testable ALU with Self-Adjacency, (c) Non-Observable ALU due to Self-Adjacency

combinational logic blocks (CLBs) must be configured into *kernels*. A kernel is a combinational block that is fed directly or indirectly by a pseudorandom pattern generator, and each output feeds either directly or indirectly a signature analyzer. A kernel $K(T_i)$ is uniquely determined by the test register T_i at its outputs. Intermediate results of response compaction can be used as test patterns. This leads to a sufficiently high fault coverage if the circuit is not random pattern resistant and all the states required for testing are reachable.

2. PROBLEM FORMULATION

Given a non-testable RTL design, expressed in a hardware description language such as VHDL, the objective is to transform it into a testable one with the least cost. We define in this context two types of designs, *synthesized* designs and *manual* designs. Synthesized designs are designs that are generated using high-level synthesis tools. On the other hand, we label manual designs, designs that are partially or fully designed by designers. Either way, if such designs are not testable, then it is important to transform those existing designs into testable ones.

This work is motivated by the need to include DFT insertion in large, high speed designs. To solve this problem, we consider test insertion in an integrated fashion with the design process as well as an after synthesis approach. We integrate the above issues within a unified graphical user interface. The problems we address in this paper are as follows. Given a register-transfer level (RTL) description of a datapath, im-

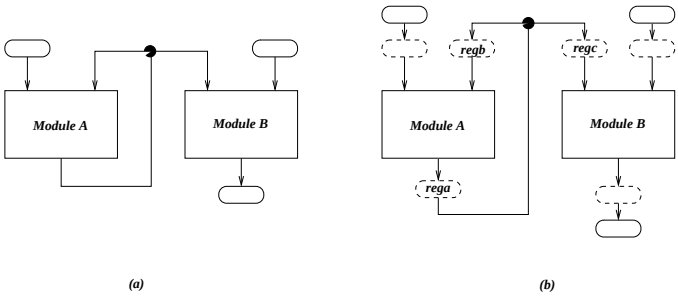

Figure 2. (a) Initial design; (b) Design after pseudo_registers insertion

prove its testability by: 1) inserting additional registers, active during test only, if necessary; 2) Interactively convert already existing registers into LFSRs so that they can be configured as TPGRs, MISRs, BILBOs, or CBILBOs during test mode.

The remainder of this paper is organized as follows. Section 3. presents the redesign process while section 4. discusses the redesign process through an example. Finally, section 5. describes the system simulation.

3. REDESIGN APPROACH

In order to transform the design into a testable one, we use the notion of structural testability which we introduced in our earlier work [8]. The key element in the data path structural testability is the Testable Functional Block shown in Figure 1 where *kernel* K_2 can be easily tested. A data path that is composed of TFBs is structurally testable. In order to transform a given datapath into a structurally testable one, we analyze its testability and enhance its observability as well as its controllability by inserting additional registers which are only active during test mode. Such registers are called test points. We solve the BIST insertion problem in two stages. In the first stage, necessary registers are inserted in the datapath so that to guarantee its structural testability. In the second stage, datapath registers configurations are explored in order to improve the datapath cost while trading-off with test time and quality.

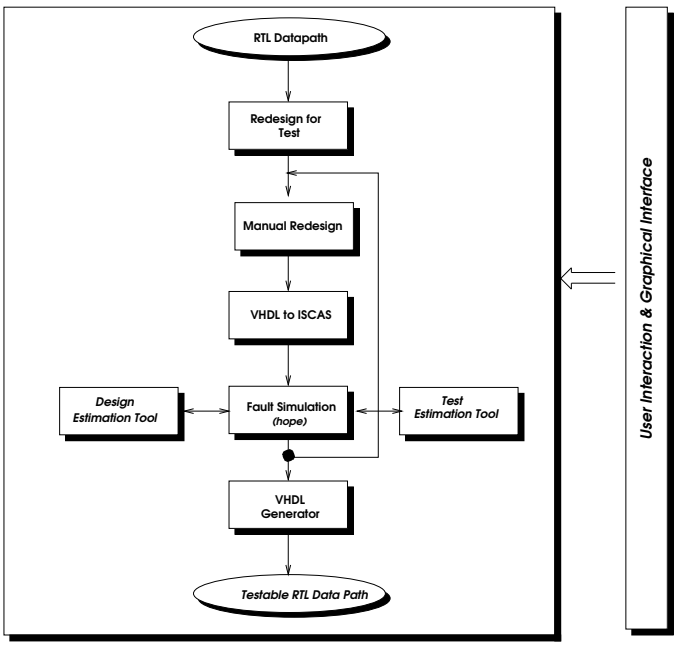

Figure 4. System Flow

3.1. Phase I: Test Points Insertion and Selection

In phase I, we transform the design into testable templates that will render the data path structurally testable. Thus, additional registers will be inserted and configured as test registers during test mode. Initially, assume all inserted registers are test registers (normal). We first modify the design structure in order to have test points. We accomplish this by inserting a pseudo-register at every module input and output ports. The idea in here to cover all the ports with at least one register, converted in the next stage into a test point. The pseudo- register will be only selected if a specific port is not covered by one register. We solve this problem using a heuristic weighted set covering problem since different test registers do have different cost. However, the problem could also be solved using ILP formulation, resulting in an optimum registers selection.

To illustrate the pseudo-registers selection process, we use the example shown in Figure 2. Note the existence of the self-

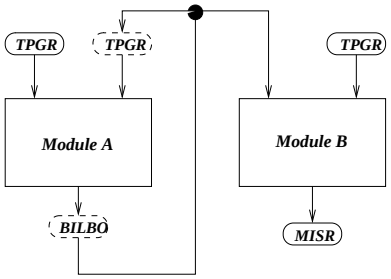

Figure 3. Structurally testable solution (Two test sessions)

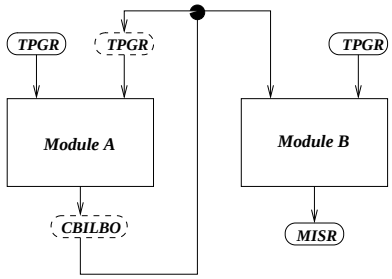

Figure 5. Structurally testable solution (One test session)

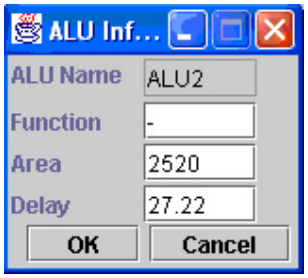

Figure 6. Modifying the ALU attributes for redesign purpose

loop around *module A* in Figure 2(a). The initial datapath is transformed into one consisting of two TFBs (Figure 3), after selecting the minimum number of registers from Figure 2(b). Note that it is possible also to select two registers to be inserted in the final solution if the datapath to be tested in one session (Figure 5). The decision in this case is left to the designer as a constraint.

Once the datapath has been ensured to be structurally testable, an initial test points assignment is chosen where all registers connected to primary inputs are configured as TPGRs, registers connected to the primary outputs are MISR, and those in between are configured as BILBOs or CBILBOs (Figures 5 and 3). Obviously, this results in a datapath characterized with a high fault coverage but which incurs additional hardware overhead and delay. In order to reduce the test hardware overhead, we can remove test points by considering modules functionality, using the test metrics [4]. The idea is to remove a TPGR if it is at the output port of a module, whose responses are random. In the same token, an MISR is removed if the faults can be propagated through an intermediate module to another MISR without loss in randomness. Other solutions maybe possible by inserting other types of test registers such as BIBLOs and CBILBOs. These registers will improve the testing time and fault coverage but on the expense of delay and area. These alternative solutions will be next interactively explored in phase II.

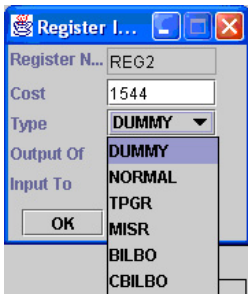

Figure 7. Modifying test registers for redesign purposes

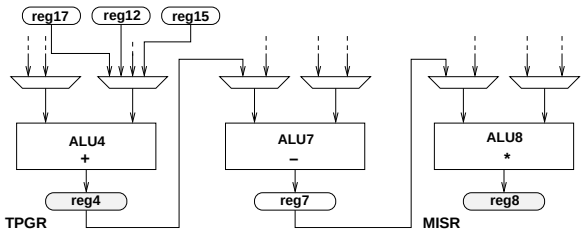

Figure 8. Initial chain

3.2. Phase II: Interactive Test Redesign

In order to further improve the design fault coverage, test area, and test time, the user may inject test points in the system by interactively interacting with a graphical user interface. The graphical user interface integrates the redesign tools in a unified environment and allows for higher designer intervention and design space exploration. Thus, the user can interactively diagnose and pin-point fault simulation weaknesses in the design by boosting test points in the design in order to improve its testability as well as its test time. Alternatively, some test points maybe removed from the design thus reducing area and power. The process is followed by a quick fault simulation that can measure the validity of the approach.

The system starts by parsing the structural VHDL datapath and a netlist is constructed. The netlist is next fed back to the GUI interface where the design is graphically displayed as shown in Figure 14. The user now can have the option of simulating the design or checking its fault coverage. In that case, the datapath is transformed into the ISCAS format and then automatically fault graded using the hope fault simulator [12]. The result of the fault simulation can be displayed on the GUI interface as well as shown in Figure 15.

The designer can now interactively modify and select test points. The system will give hints based on the resulting fault simulation which is estimated based on the hope fault simulator as well as on the number of test sessions in addition to the design area and performance.

In order to illustrate the redesign process we use the example in Figure 8 where the fault coverage can be increased by inserting additional test points *(reg7)* as shown in Figure 9. Likewise, it is also possible to remove unnecessary test points while maintaining a specific, acceptable fault coverage. Since test time is also an important factor, the test point selection will also aim at minimizing the test time by increasing the number of concurrent test sessions. This will be done through test scheduling.

4. TEST TRADEOFFS ILLUSTRATION USING AN EXAMPLE

In order to illustrate our tradeoff scheme, we use the DFG of the TRIGO example, shown in Figure 10. The TRIGO DFG computes the Taylor series for the trigonometric func-

tions $sin(t/T)$ and $cos(t/T)$ concurrently. All the operations for this example exist in our library ({+},{*},{/}, and also some of their combinations. The test hardware overhead was significantly reduced (by 17.96 %) after applying our test point selections. We discuss next some important aspects of our tradeoff model concerning tradeoffs coverage.

We have experimented with the fault coverage of several design styles for the trigo example versus the number of test patterns as shown in Figure 11. We notice that there is a measurable difference in fault coverage quality between the *BILBO* and the *test point selection option*, in favor of the *BILBO*. This difference is of about 2% but it should be measured against the area increase required by the *BILBO* designs when making tradeoff decisions.

We also note another interesting tradeoff measure that exists between *test area overhead* and *test time* or *fault coverage*. To be specific, our tradeoff scheme has the capability to "inject" observable points (MISRs) during the design process for the benefit of increasing the fault coverage (or reducing the number of test patterns). Thus, we can generate designs between the *BILBO* and the *test point selection* options by performing tradeoffs between test area overhead and fault coverage. This tradeoff situation is illustrated in Figure 11 for the trigo circuit. By injecting an observable register we can "break" a chain of multiplier-adder in the circuit. This chain is used to bypass test patterns from a controllable point (multiplier input) to an observable point (adder output). Breaking such chains improves the fault coverage of the circuit at the expense of increasing the circuit overhead; however, this increase is not very large. These tradeoffs are depicted in Figure 11 by shifting the test pattern/fault coverage curve of *the test points selection option* to the "right" depending on the test register injections we cause. We actually illustrate in Figure 11 three fault coverage/test pattern curves corresponding to one, two and three test register injections. From the fault coverage plots of Figure 11, we can derive tradeoff relationships between test hardware and fault coverage/test time. This relationship is shown in Figure 12 which depicts the normalized test hardware overhead versus the fault coverage assuming constant test patterns for each curve. The data points in each curve were derived by injecting a test register from one data point to the next one.

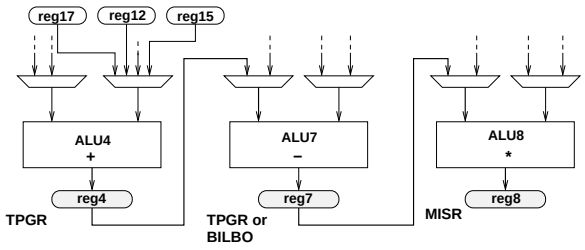

Figure 9. Broken chain with additional test points insertion

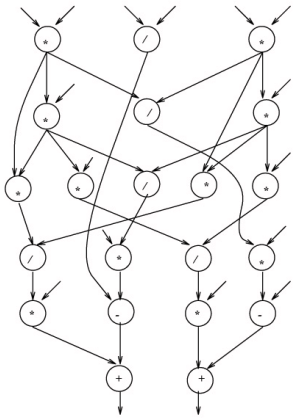

Figure 10. Scheduled DFG for the trigo example

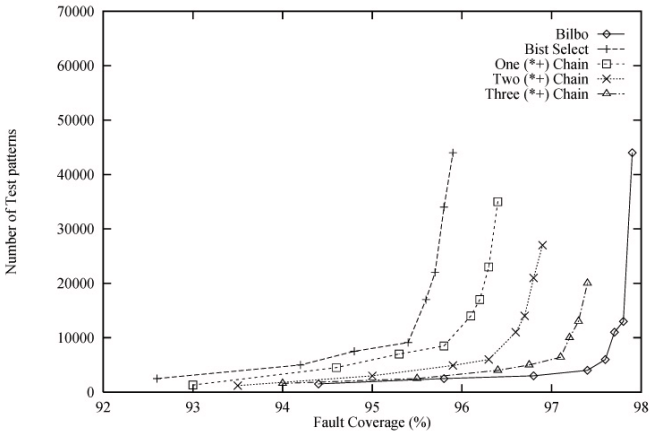

Figure 11. Fault coverage for the trigo example after injecting one, two, and three additional test points

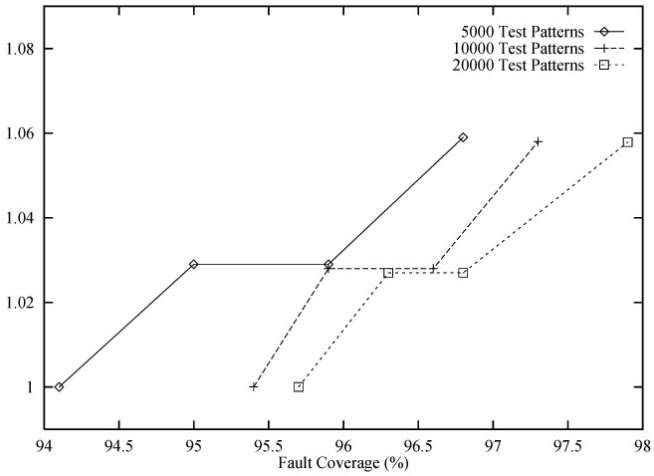

Figure 12. Relative relationship between test overhead and test time for the trigo example

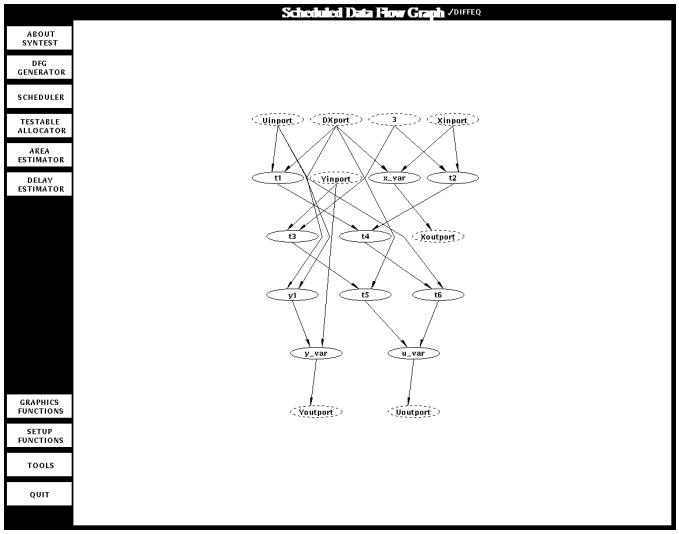

Figure 13. Differential equation schedule

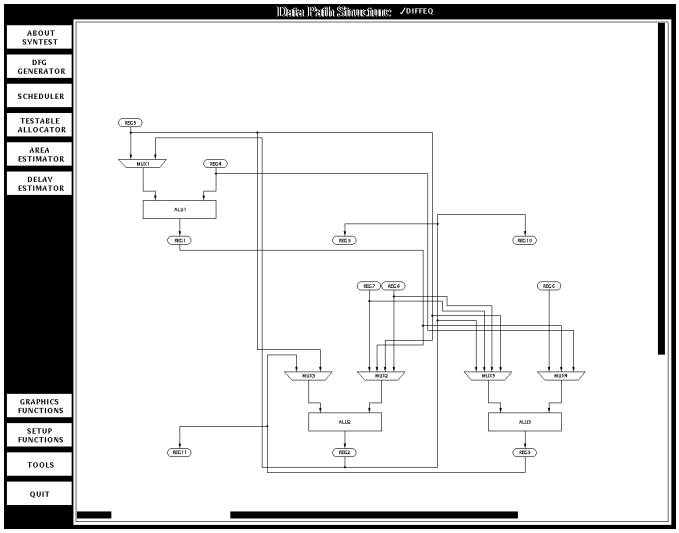

Figure 14. Synthesized differential equation data path

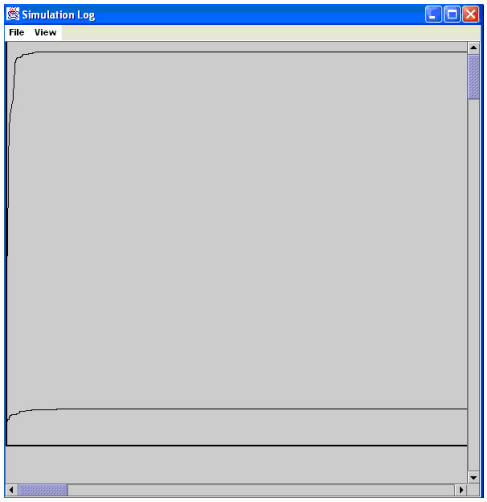

Figure 15. Fault Simulation Results Generated by our System

5. THE DIFFERENTIAL EQUATION EXAMPLE SIMULATION

The HAL differential equation example solves the following second order differential equation: $y'' + 3xy' + 3y = 0$. The first step in the design process is to capture the initial behavior using VHDL, done using a text-editor or graphically. The VHDL code is parsed and an intermediate code is generated and passed to the scheduler. The code sequence generated by the parser is next scheduled in order to exploit any parallelism that may exist under resource and time constraints. The example was scheduled in four time steps with the assumption that only two adders, two multipliers, and one subtractor are available in the library. This will affect the operations concurrency in the schedule. The resulting schedule is shown in Figure **??**. In the next step, the allocator is executed in order to allocate hardware resources for our example. The allocation process is more interesting since many design styles and solutions possibilities can be explored using the various *design and test options* available in the system. The allocator results in designs with test points selected. Designs will vary in area size and delay depending on the options used. The most expensive design is based on the BILBO methodology with the cheapest being designs without test considerations. Design comparisons for this example are shown in Table 1.

Option	Area (λ^2)	Routing Factor	# Standard Cells	Transistors Number
Normal	3,505,230	68%	309	4403
BILBO	5,628,720	75%	536	6739

Table 1. Differential Equation Resulus Using Synopsys

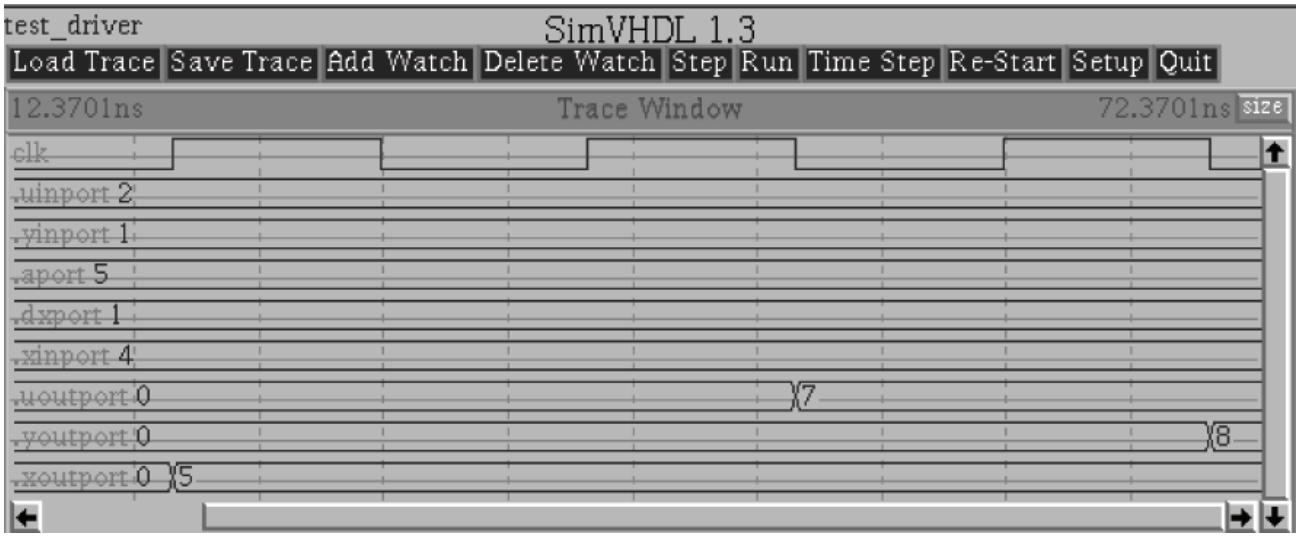

Figure 16. Differential Equations Simulation Results

REFERENCES

[1] L. Avra, "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths," *in Proc. ITC*, pp. 463-472, 1991.

[2] P. Bukovjan, L. Ducerf-Bourbon, M. Marzouki, "Cost/Quality Trade-Off in Synthesis for BIST," *JETTA,* Vol. 17, pp. 109-119, 2001.

[3] M.L. Bushnell, V.D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed Signal VLS Circuits*, Kluwer Academic Publishers, Boston, 2000.

[4] S. Chiu, C. Papachristou, "A DFT Scheme with Applications to Data Path Synthesis," *Proc. DAC*, 1991.

[5] G. Craig, C. Kime, and K. Saluja, "Test Scheduling and Control for VLSI Built-In Self-Test," *IEEE Trans. on Computers*, Vol. C-37, pp. 1099-1109, 1988.

[6] M. Dhodhi, F. Hielscher, R. Storer, J. Bhasker, "Datapath Synthesis Using a Problem-Space Genetic Algorithm," *IEEE Trans. on CAD*, Volume 14, pp. 934–944, August 1995.

[7] M. Garey, D. S. Johnson, *Computer and Intractability*, W. H. Freeman, 1979.

[8] H. Harmanani, C. Papachristou, "An Improved 'Method for RTL Synthesis with Testability Trade-Offs," *in Proc. of the ICCAD,* pp. 30-37, 1993.

[9] H. Harmanani, M. Kodeih, "Concurrent BIST Cost Estimation During Data Path Allocation," *in Proc. of the First NEWCAS,* pp. 57-60, 2003.

[10] C.L. Hudson, G. Peterson, "Parallel Self-Test With Pseudo-Random Test Patterns," *Proc. ITC*, 1987.

[11] K. Kim, J. Tront, D. Ha, "Automatic Insertion of BIST Hardware Using VHDL," *Proc. DAC*, 1988.

[12] H. Lee and D. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *Proc. 29th DAC*, June 1992.

[13] I. Parulkar, S. Gupta and M. Breuer, "Scheduling and Module Assignment for Reducing BIST Resources," *in Proc. DATE 98.*

[14] L.T. Wang, E.J. McCluskey, "Concurrent Built-In Logic Block Observer," in *Proc. ISCAS'86,* pp. 1054-1057.