A Method for Redesign for Testability at the RT Level

Haidar Harmanani and Salam Harfoush
Computer Engineering & Science Deps.
The Lebanese American University
Byblos, Lebanon

Abstract

A new method of redesign for testability at the Register-Transfer Level (RTL) is proposed. The method identifies hard to test parts of a an RTL design synthesized either manually or automatically using high-level synthesis tools. The design is modified by inserting additional test registers followed by a test selection process. During the selection process, two test metrics are used in order to minimize test overhead. Finally, test scheduling is performed so that to minimize the overall test time and the number of test sessions. The system outputs a VHDL description of a testable data path along with a test plan.

1 Introduction

The complexity of VLSI circuitry has complicated the design and test of digital circuits. *High-level synthesis* [4] has emerged as a good approach for top down design methodology. Though some recent works have integrated [2, 6, 7] the design and test process, most researches have ignored testability consideration at the system level. Add to that the numerous designs that were *manually* synthesized without testing consideration. In this paper, we refer to *manual designs* as designs that were not synthesized using high-level synthesis tools.

Design for testability (DFT) [1] techniques emerged as a solution that aims at efficient and cost effective testing by enhancing the controllability and observability of the circuit under test. Within DFT, Built-In Self-Test (BIST) was proposed with test generation and response analysis occur on-chip. The advantages fo BIST is that designs can be "tested per clock." Thus, potentially enhancing test application time as well as delay testing. One of the main disadvantages of BIST is that it requires a higher hardware overhead.

One of the difficulties in implementing BIST techniques is the register self-adjacency problem[1], arising due to structures similar to the one shown in Figure 1(a). In this circuit, it is not possible to assign *R2* as both a pattern generator and a signature analyzer at the same time. Hudson [8] showed that when self-adjacent registers are configured as test pattern generators, the additional feedback that made the register self-adjacent can greatly reduce its ability to retain error information as a signature analyzer.

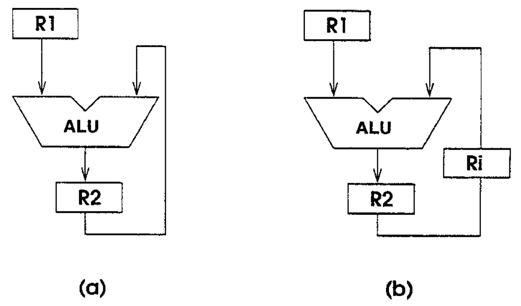

Figure 1: (a) Non-Observable ALU due to Self-Adjacency, (b) Testable ALU after test insertion

1.1 Problem Description and Significance

The problem we address in this paper is as follows: Given an RTL description of a datapath, the purpose of the redesign for testability method is to improve its testability by: 1) inserting additional registers, active during test only, if necessary; 2) schedule the resulting structure into a minimal number of test sessions so that to reduce the overall test time.

In order to reduce test penalty and ensure the datapath structural testability, it is necessary to automate the BIST insertion process. The BIST insertion problem is solved in two stages. In the first stage, necessary registers are inserted in the datapath so that to guarantee the datapath structural testability. In the second stage, datapath registers configurations are explored in order to improve the

This work was supported in part by a grant from the Lebanese Council for Scientifc Research.

[1]A register is self-adjacent if an output of that register feeds through combinational logic and back into itself.

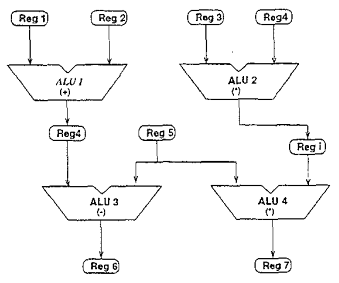

Figure 2: Simple Data Path Example

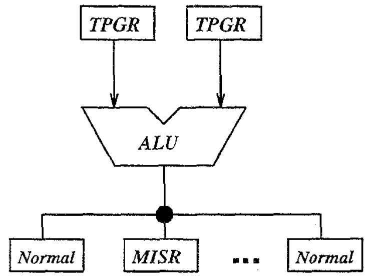

Figure 3: Extended Testable Functional Block

datapath cost while trading-off with test time and quality. The resulting structure is finally synthesized in VHDL, including test structures, and fed to a logic synthesis tool.

This paper is organized as follows: in section 2 we describe the test insertion process. Section 3 describes the selection process while section 4 describes our scheduling approach. Finally, results are presented and discussed in 5.

2 Registers Insertion for BIST

2.1 Basic Idea

Assume we have the example shown in Figure 2. In order to be able to apply the BIST methodology to this example, *regi* to be inserted in order to compress test patterns for ALU2 and generate random patterns for ALU4. If we assume that ALU1 can generate random patterns that can propagate to ALU3, then *reg4* need not be a pattern generator for ALU3. Furthermore, if we assume that ALU3 is transparent enough such that *reg6* can catch faults in ALU1, then *reg4* need not be a signature analyzer for ALU1. *Reg4* woud have been configured as a BILBO[2] register otheriwse. Using a similar reasoning, *regi* is configured as an MISR, while otherwise, it would have been configured as a BILBO. In addition to decreasing the area overhead, test metrics can decrease the number of test sessions. For example, ALU2 and ALU4 are scheduled in the same test session while otherwise they would be scheduled in different test sessions.

2.2 Approach

In order to guarantee the datapath structural testability, we use the notion of structural testability which we defined in [7]. The key element in the datapath structural testability is the Testable Functional Block shown in Figure 1. A datapath composed of TFBs is structurally testable [7].

[2]A BILBO is a test register that can be configured as an MISR or a TPGR, but in different test sessions

As illustrated in the previous section, we may need to insert additional registers in order to transform a given datapath into a structurally testable one. The rational is to modify the datapath in order to create TFBs. The key element is to break any self-loops which are feeding a module output back to its input [8] by inserting additional registers.

The optimization problem is formulated as follows. We insert a pseudo-register[3] at every module input and output ports. The idea is to cover all datapath ports with at least one register, to be 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 an ILP formulation.

3 BIST Test Points Selection

Once the structure has been ensured to be structurally testable, we choose an initial test points assignment 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. Obviously, this results in a datapath characterized with a high fault coverage but which incurs additional hardware overhead and delay. There are two conditions that should be satisfied:

1. The TPGRs at the input ports of a single ALU cannot be the same due to correlation problems.
2. A TPGR cannot be an MISR for the same ALU in the same test session in order to avoid the self-adjacency problem.

In order to reduce the test hardware overhead, we remove test points by considering modules functionality, using the test metrics developed in [3]. Thus,

[3]A pseudo-register is a register that may or may not be inserted in the final datapath

we 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 a loss in randomness. For example in Figure 2, ALU3 is transparent; therefore we can use the same register, *reg6* as an MISR for ALU1 as well as for ALU3. On the other hand, if an ALU is random, then its TPGRs may be selected as a TPGR for other ALUs. In Figure 2, ALU1 is random, and therefore its TPGRs, namely *reg1* and *reg2* may be selected as TPGRs of ALU3 connected to ALU1 at its output port. This feature helps minimizing the number of test points selected while ensuring the circuit testability.

The above process will result in two possible extreme solutions. However, it is possible to explore more design alternatives using additional test points. The approach is as follows. Each module is simulated using fault simulation and the corresponding fault coverage is recorded in the technology library. We next simulate chain of components. For example, we simulate an *(adder, multiplier)* chain or an *(adder, subtractor)* chain. The next question becomes one of what modification of test points on these chains should be considered so that to achieve the desired fault coverage, specified as a user constraint. We use a heuristic approach to solve this problem, based on a branch and bound method.

3.1 The Scheduling Process

The final step is the scheduling of different datapath components into different test sessions. The goal of this step is to minimize the number of test sessions by maximizing the number of ALUs tested in the same test session. However, there are conditions which restrict two ALUs from being tested at the same time. Such ALUs have to be tested in different test cycles. These conditions are:

- If two ALUs have the same MISR, then they cannot be tested at the same time, they have to be applied to different test sessions.
- If an ALUs MISR is another ALUs TPGR then these two ALUs cannot be tested at the same time, unless the register in concern is a CBILBO which is not used at this stage in our system. Therefore, these two ALUs have to be scheduled into two different test sessions.

The scheduling process has been proven to be NP-complete; therefore, a heuristic has been used to derive a sub-optimal solution. The scheduling process is divided into two main steps that resolve respectively the first and the second condition. In the first step, ALUs that have the same MISR are assigned to different test sessions. Each ALU is assigned a weight that is equal to the number of times its MISR is used by other ALUs. The second step satisfies the second condition. Thus, every ALUs TPGRs are compared to every other ALU's MISR in the same test session. If they use the same register, then one of the two ALUs that has the minimal weight is moved to another test session. The running time of the scheduling algorithm is in the worst case equal to $O(a^2)$ where a is the number of ALUs in the input circuit.

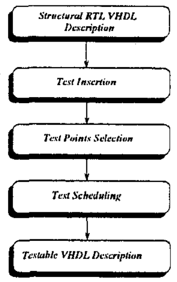

Figure 4: ReTest System Description

4 Results

In order to validate our Redesign for Testability (ReTest) approach, we ran some benchmark examples generated using High-Level Synthesis tools. In what follows, we explain the experimental procedure and then we discuss a couple of designs.

4.1 Experimental Procedure

In order to validate our system, we captured designs that were generated using high-level synthesis tools in VHDL. We then translate the designs from VHDL to ISCAS89 format. The designs were then fault graded, before and after the test insertion, using the HOPE simulator [5]. We fed the simulator random pattersn that were generated using the LFSR compiler in COMPASS. Three designs were attempted for this paper, all were derived from the literature.

4.1.1 Example 1: ARYL and LYRA

The first example that we used was synthesized from behavioral description using ARYL and LYRA [9]. Only one register was inserted for this example in order to improve the testability of one ALU. The datapath was scheduled in three test sessions. The fault coverage of the redesigned circuit improved to 98.71%, from 87.79%, an improvement of 10.919%. The fault simulation time improved from 9.733 secs

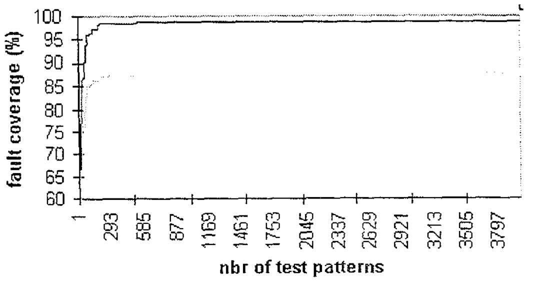

Figure 5: Fault simulation results using ARYL and LYRA

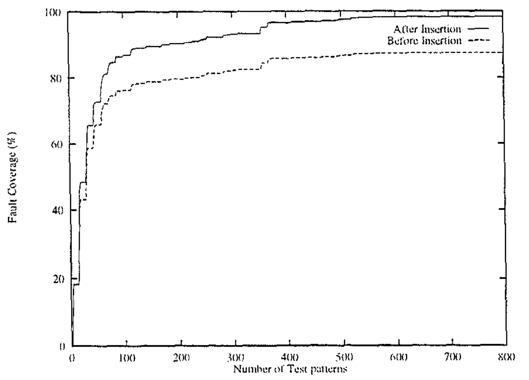

Figure 6: Differential Equations fault simulation results using [11]

to 6.667 secs. The example was scheduled in three test sessions. Fault simulation results are shown in Figure 5.

4.1.2 Examples 2 and 3: Differential Equation

The second example that we attempted was the HAL differential equation popularized by Paulin [10]. We used two designs generated by [3] and [11].

One register was inserted in the first design; thus, improving its testability from 58.53% to 98.11%. The simulation time also improved from 39.7 secs to 4.41 secs. The example was scheduled in four test sessions. For the second design, one register was also inserted. The fault coverage improved from 92.16% to 96.41%; however, the fault simulation time did not improve by much (only 0.6 secs). The example was scheduled in three test sessions. Fault simulation results are shown in Figure 6 and 7

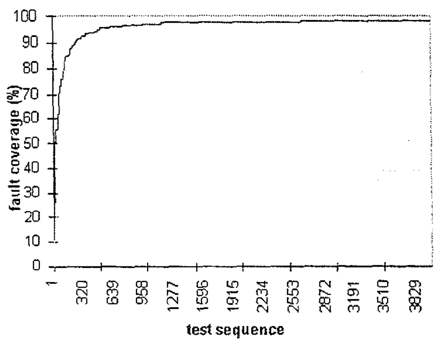

Figure 7: Differential Equations fault simulation results using [3]

References

[1] M. Abramovici, M. Breuer, A. Friedman, *Digital Systems Testing and Testable Designs,* Computer Science Press, 1990.

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

[3] S. Chiu, C. Papachristou, "A Design for Testability Scheme with Applications to Data Path Synthesis," *Proc. 28th DAC,* June 1991.

[4] G. De Micheli, *Synthesis and Optimization of Digital Circuits,* McGraw Hill, 1994.

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

[6] C. Gebotys, M. Elmasri, "VLSI Design Synthesis with Testability," *Proc. 25th DAC,* June 1988, pp. 16-21.

[7] H. Harmanani, C. Papachristou, "An Improved Method for RTL Synthesis with Testability Trade-Offs,' *Proc. ICCAD,* 1993.

[8] C.L. Hudson, G.D. Peterson, "Parallel Self-Test With Pseudo-Random Test Patterns," *Proc. ITC,* pp. 954-971, Sept. 1987.

[9] C. Huang, et. al, "Data Path Allocation Based on Bipartite Weighted Matching," *Proc. 27th DAC,* 1990.

[10] P. Paulin, J.P. Knight, "Forced-Directed Scheduling for the Behavioral Synthesis of ASIC's", *IEEE Trans. CAD,* June 1989.

[11] B. Pangrle, "Splicer: A Heuristic Approach to Connectvity Binding," *Proc. 25th DAC,* 1988.