TESTCAD: A Verified Education Toolset for a Course in Digital Testing

Spencer Millican,
Auburn University
millican@auburn.edu

Kewal Saluja
University of Wisconsin - Madison
saluja@engr.wisc.edu

Abstract – Education in CAD at the undergraduate and graduate level is needed to train exceptional VLSI engineers. This is especially true for DFT engineers, whose skills are in high demand with increasing circuit reliability requirements. Acquiring access to CAD and DFT tools for education may not be viable to educational institutions, and teaching DFT without such tools will yield marginal knowledge retention and fewer applicable skills. Managing commercial tools and licenses also requires resources which are better suited to other educational or research needs. This article presents the open-source TESTCAD Educational Toolset, which implements several DFT functions (ATPG, fault simulation, and test compaction) on circuits. The tools also include support programs which enforce academic honesty and rigor, thereby increasing the quality of the tools in an educational environment. The command-line and open-source implementation of the tools allows for easy expansion for nuanced educational environments, and the tools can be modified for research experiments as needed.

I. Introduction

Education in electronic design automation (EDA) and is in high demand in order to meet increasing performance requirements for low-cost electronics. The rising demand of system-on-chips (SoCs) and "internet of things" (IoT) devices is decreasing time-to-market requirements, which in turn puts a burden on circuit designers. Designers have overcome this by using EDA tools to automate and streamline their design flows. Although these tools give an opportunity to significantly decrease development time, this opportunity can only be leveraged if the designer is knowledgeable in the use of the tool. For this reason, education in the use and theory of EDA tools at colleges and universities is paramount in order to keep the circuit industry competitive.

DFT (design for testability) is in particular demand given the increasing demand for reliable electronics. With electronics being deployed in more life-critical applications (e.g., self-driving cars), the consequences of deploying defective circuits are amplified. Developing efficient tests keeps device manufacturing costs low and prevents the release of defective circuits to the market. Several EDA tools are part of the DFT process (test hardware insertion, fault simulation, test pattern generation, etc.), but all these tools must be understood by the designer before they can be effectively used.

EDA tools are required in order to educate students in their theory and application, but industry tools are not always suited or available for education. Although classroom instruction is a vital part of engineering education, hands-on laboratory experience is needed to reinforce what is learned in the classroom. Many companies offer their tools under education licenses, but their use has drawbacks. First, these licenses may not be available to all institutions that apply for them, and managing these licenses requires administrative resources from the university which may not be available. Second, using these tools requires the systems they are installed on to be configured correctly, which in turn requires additional administrative resources. Third, the use of these tools often risks educating "the steps of the tools" as opposed to the theory behind the tools.

This article presents the TESTCAD Educational Toolset. This toolset has been used extensively at UW – Madison's course "ECE 553: Testing and Testable Design of Digital Systems" for use in homework and to implement a course final project. This toolset consists of open-source command-line tools which implement common DFT tasks. The toolset also contains programs which allow the tools to be used effectively in an educational environment where academic rigor is of importance.

II. TESTCAD Inputs and Outputs

The TESTCAD tools are command-line tools which take ASCII text as their inputs and outputs. Programs pass their inputs and outputs either through command-line arguments or through standard streams. This environment allows students

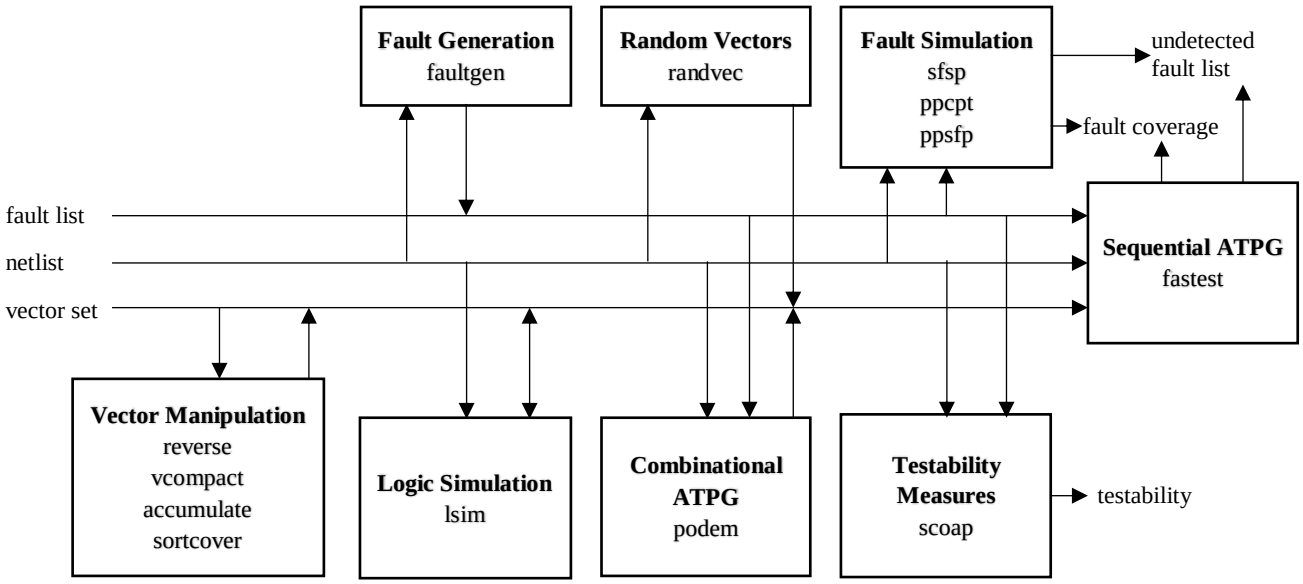

Figure 1: Here, the interaction between different input and output files is illustrated.

to learn how to navigate programs in a Unix environment and how to build complex CAD flows through scripting.

The relationship between shared input and output types is illustrated in Figure 1and are as follows:

Netlist: This file or stream contains a netlist representation of a circuit in a netlist file format. This netlist can optionally contain sequential circuit elements (i.e., DFFs).

Fault list: This file or stream contains a list of faults in a fault list file format. A fault file is required for fault simulation. This file can be the output of a program which provides a list of undetected faults.

Vector set: This file or stream is composed of a list of test vectors in a vector set file format. It can be generated manually, by random test pattern generation, or by test pattern generators. It can be used by both fault simulators or logic simulators. When used by logic or fault simulation, the (faulty) output of a circuit can be included in a new vector set.

Fault coverage: Fault simulation programs will output the percentage of detected faults in a circuit. This will be presented only in terms of all faults simulated, and therefore any fault equivalence collapsing must be done by reducing the fault list given through *faultgen*.

Testability: *scoap* can attach testability measures (CC0, CC1, CO) to every net in a circuit as well as calculate the probability a list of faults will be detected using random vectors.

III. DFT TOOLS

The following command-line tools are available for generating and manipulating fault lists and vector sets.

randvec – *Random vector generator*: This program has two uses. First, it generates random vectors for a given circuit. Second, it fills unspecified (X) entries with 0's and 1's for a given list of test vectors. The proportion of 1s and 0s produced in X bit positions can be specified.

faultgen – *Fault list generation*: This program generates a list of stuck-at faults for a given circuit. Optional parameters will reduce the list of faults through gate-level fault equivalence.

podem – *Test pattern generator*: This program generates test vectors for a given circuit and a given fault list using the PODEM [1] algorithm.

lsim – *Logic Simulator*: This program produces a set of output vectors (circuit outputs) for a given set of test vectors (circuit inputs) and a given circuit. The output will be a modified version of the input vector file or stream with circuit output values added.

sfsp – *Single fault, single pattern propagation fault simulator*: This program simulates a given circuit for a given list of faults and list of test vectors. It reports undetected faults and fault coverage statistics. Undetected faults are in the fault list file format in order to be used in subsequent test generation or fault simulation. Since it employs a single fault, single pattern propagation method, it is slower than *ppcpt*, but it is useful for simulating partially-specified (three-valued) test vectors. To avoid excessive simulation time, it should not be used to simulate a large number of vectors or a large number of faults.

ppcpt – *Parallel pattern critical path tracing fault simulator*: This program simulates a given circuit for a given list of faults and a given list of test vectors. It reports undetected faults and fault coverage statistics. Since it is a parallel pattern simulator, it is best suited for simulating 32 or more test vectors.

ppsfp – *Parallel pattern single fault propagation fault simulator*: This program simulates a given circuit for a given list of faults and a given list of test vectors. It reports undetected faults and fault coverage statistics. Since it is a parallel pattern simulator, it is best suited for a vector set file containing 32 or more test vectors.

scoap – *Testability evaluator*: This program computes a number of different testability values for lines within a circuit and compute statistics for the entire circuit using the SCOAP [2] algorithm. The values computed are controllability, observability and testability of every line and fault.

reverse, vcompact – *Manipulating test vectors*: These tools manipulate a list of vectors. They can reverse the order of a vector set and merge two identical vectors with available don't-care (X) bits.

accumulate, sortcover – *Compacting test vectors through simulation*: These tools perform fault simulation (using *sfsp*) for every vector in a file. The first tool will remove a vector if no additional faults are detected, and the second tool will sort vectors by how many faults they detect.

fastest – *A sequential test pattern generator*: This program generates a sequence of test vectors for a given sequential circuit and a given fault list using the FASTEST [3] algorithm.

IV. Course Material Generation Tools

To allow these tools to be used in an education environment, a supplemental tool is available to implement a DFT workflow while enforcing academic honesty.

cryptnet – *A circuit scrambler program*: This program takes a netlist file as an input and "scrambles" the circuit while keeping the function of the circuit intact. This is done by renaming nets in the circuit and modifying circuit gates in such a way that the function of the circuit is not modified.

This program can be used in two ways. First, it can create separate copies of the same circuit such that "answers" (fault coverage, diagnosed faults, etc.) regarding a circuit are unique to that circuit. Doing this will make correct student assignment responses unique. Second, this program can be used to make "identical" copies of the same circuit while obfuscating their implementation. This allows for multiple good and faulty "chips" to be made for simulation based off of the same "good" and "faulty" chip. This allows for a manufacturing test environment to be simulated where students must test for faulty chips amongst good once and diagnose the fault in faulty chips.

V. Classroom Exercises

Many classroom exercises are possible using the tools provided. The following are a few examples.

Fault diagnosis: A good chip can be modified with a fault and then scrambled using *cryptnet*. The student then must diagnose which fault was injected into the chip by applying fault simulated vectors, observing which do (not) detect the fault, and using the process of elimination.

Fault simulation time experimentation: With many different methods of performing fault simulation, the student can experiment with the benefits and detriments of the different methods under different simulation environments

Test set minimization: For a given chip, the student must create a vector set of a minimum size which detects as many faults in the circuit as possible. The student can be encouraged to

implement a realistic ATPG flow (e.g., use random vectors and then create PODEM top-off patterns). In a class project setting, this can be implemented by giving each student the same circuit after scrambling with *cryptnet* and then grading based on who achieved the smallest number of vectors or the highest fault coverage.

VI. Modification & Research

Source code for the tools is available for modification. Each program and script has its own code with some common functions shared amongst multiple programs. The source code for these tools is written in C and Perl.

With the availability of the source code, it is the authors' intention to make these programs modifiable to allow for expedited research projects with fewer research resources. Given many DFT research projects use common DFT flows, research can be facilitated by modifying or replacing the appropriate programs in a flow and observing the effect this has on fault coverage, simulation time, etc. For example, if a new ATPG method is proposed to increase fault coverage, the "base flow" can be made using *podem* for ATPG, and the new proposed flow can replace this program with a modified version.

VII. Conclusions

The tools presented here give course instructors the opportunity to introduce applied DFT education with the use of open-source tools. By not relying on industrial tools requiring licensing procedures, more students will have access to the tools and education can focus on the theory of DFT as opposed to the operation of specific industrial tools.

VIII. References

[1] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. C–30, no. 3, pp. 215–222, Mar. 1981.

[2] L. H. Goldstein and E. L. Thigpen, "SCOAP: Sandia Controllability/Observability Analysis Program," in *17th Design Automation Conference*, 1980, pp. 190–196.

[3] T. P. Kelsey, K. K. Saluja, and S. Y. Lee, "An efficient algorithm for sequential circuit test generation," *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1361–1371, Nov. 1993.