



Ain Shams University

Faculty of Engineering

Computer and Systems Department

Hardware Based Automatic Test Pattern Generation

A Thesis

Submitted in partial fulfillment for the requirements of Master of
Science degree in Computer Engineering

Submitted by:

Hany Ismail Kashif

B.Sc. of Computer Engineering

(Computer and Systems Department)

Ain Shams University, 2007.

Supervised by:

Prof. Dr. Ashraf Mohamed El-Farghaly Salem

Dr. Mona Ahmed Fahmy

Dr. Ahmed Hassan Mohamed

Cairo 2010

Statement

This dissertation is submitted to Ain Shams University for the degree of Master of Science in Electrical Engineering (Computer and Systems Engineering).

The work included in this thesis was carried out by the author at the Computer and Systems Engineering Department, Faculty of Engineering, Ain Shams University, Cairo, Egypt.

No part of this thesis was submitted for a degree or a qualification at any other university or institution.

Name: Hany Ismail Kashif Mohamed Hany

Date: June 19, 2010

Acknowledgments

All praise is due to Allah, Most Gracious Most Merciful, the Lord of the Worlds, Who taught man what he knew not. I would like to thank God Almighty for bestowing upon me the chance, strength and ability to complete this work.

I would like to thank my supervisors: Prof. Ashraf Salem, Dr. Mona Fahmy and Dr. Ahmed Hassan for their direction, assistance and guidance. I would also like to thank Mona Safar for her suggestions and recommendations which helped me a great deal in fulfilling this thesis.

I as well need to thank my parents who have helped me in many ways. And, finally, words are not enough to express the thanks and gratitude I owe to my wife for her continuous encouragement and support.

Abstract

Automatic Test Pattern Generation (ATPG) is a well known NP-complete problem and has its high importance in digital testing in the Electronic Design Automation (EDA) industry. Accordingly, several types of algorithms have been developed to solve the Automatic Test Pattern Generation (ATPG) problem. Boolean Satisfiability (SAT) techniques have been used to build ATPG algorithms and have had various increments lately including learning and conflict analysis. Much of the performance improvement achieved by state-of-the-art SAT solvers is related to the implementation of conflict analysis which enables the solver to perform non-chronological conflict based backjumping.

Within the context of this thesis, we propose a novel Selective Conflict Directed Jumping (SCDJ) approach for exploring the search space of a given SAT instance. Basically, the proposed algorithm does not need to free the variables that have been assigned in previous decisions made until a conflict has been detected. These decisions are selectively used after conflict analysis to jump deeper into the search space. The proposed Selective Conflict Directed Jumping (SCDJ) scheme is integrated into the latest release of the zChaff SAT-solver and is tested against instances from the DIMACS benchmarks suite. The SCDJ algorithm is also integrated in an ATPG system (SIS) to prove its capabilities in generating test patterns. A hardware-based ATPG solver has been implemented as well using an existing hardware-based Conflict Directed Jumping algorithm through integrating it with SIS.

Contents

| | |
|---|------------|
| Table of contents | II |
| List of figures | VII |
| List of tables | IX |
| List of abbreviations | XI |
| 1 Introduction | 1 |
| 1.1 Structural vs Functional Testing | 2 |
| 1.2 Thesis Problem Statement | 2 |
| 1.3 Thesis Organization | 3 |
| 2 Automatic Test Pattern Generation | 5 |
| 2.1 Faults and Fault Modeling | 6 |
| 2.1.1 Single Stuck-at Faults | 7 |
| 2.2 Fault Propagation | 9 |
| 2.3 Automatic Test Pattern Generation | 10 |
| 2.4 Fault Collapsing | 11 |

| | | |
|----------|---|-----------|
| 2.5 | Fault Simulation | 15 |
| 2.6 | Fault Compaction | 18 |
| 2.7 | ATPG Algorithms | 19 |
| 2.7.1 | D-Algorithm | 19 |
| 2.7.2 | PODEM | 21 |
| 2.7.3 | FAN | 25 |
| 2.7.4 | TOPS | 27 |
| 2.7.5 | SOCRATES | 28 |
| 2.7.6 | EST | 28 |
| 2.7.7 | Recursive Learning ATPG | 29 |
| 2.7.8 | BDD-Based ATPG Algorithms | 29 |
| 2.8 | Boolean Satisfiability and Implication Graph ATPG | 29 |
| 3 | SAT-Based ATPG | 31 |
| 3.1 | Boolean Satisfiability (SAT) | 32 |
| 3.1.1 | SAT Algorithms | 33 |
| | Incomplete Algorithms | 33 |
| | Complete Algorithms | 33 |
| 3.1.2 | Basic SAT Solver | 34 |
| 3.2 | SAT Formulas | 37 |
| | Creating good circuit formula | 39 |
| | Creating faulty circuit formula | 41 |
| | Creating circuits XOR formula | 41 |
| 3.3 | SAT-based ATPG Algorithms | 43 |
| 3.3.1 | NEMESIS | 43 |
| 3.3.2 | TEGUS | 45 |

| | | |
|----------|---|------------|
| 3.3.3 | GRASP | 47 |
| 3.3.4 | TG-GRASP | 48 |
| 3.3.5 | CGRASP | 50 |
| 3.3.6 | TIP | 51 |
| 3.3.7 | Chaff | 52 |
| 3.3.8 | PASSAT | 53 |
| 3.3.9 | MULTI CODCs | 54 |
| 4 | zChaff SAT Solver | 57 |
| 4.1 | Chaff Algorithm | 58 |
| 4.1.1 | Overview | 58 |
| 4.1.2 | Decision Heuristics | 62 |
| 4.1.3 | Deduction and Implication | 63 |
| 4.1.4 | Conflict Analysis and Learning | 68 |
| 4.2 | zChaff Software | 72 |
| 4.2.1 | Input Format | 72 |
| 4.2.2 | Operation | 74 |
| 4.2.3 | Output | 74 |
| 5 | Selective Conflict Directed Jumping | 77 |
| 5.1 | Selective Conflict Directed Jumping Algorithm | 78 |
| 5.2 | Experimental Results | 84 |
| 6 | SCDJ Based ATPG | 115 |
| 6.1 | SIS | 116 |
| 6.2 | ATPG Using SCDJ Algorithm | 119 |
| 6.3 | Hardware Based SAT Solver | 121 |

| | | |
|----------|--|------------|
| 6.3.1 | Reconfigurable SAT solvers | 123 |
| 6.3.2 | Application Specific Conflict Diagnosis Based SAT Solver | 124 |
| 6.3.3 | Integrating the FPGA-based SAT Solver with SIS | 126 |
| 7 | Conclusion | 129 |
| 7.1 | Future Work | 130 |
| | References | 132 |

List of Figures

| | | |
|------|--|----|
| 2.1 | Circuit fault | 6 |
| 2.2 | An example of a single stuck-at fault | 8 |
| 2.3 | Equivalent faults for logic gates | 12 |
| 2.4 | Un-collapsed faults | 12 |
| 2.5 | Equivalence collapsed faults | 13 |
| 2.6 | Dominance fault collapsing | 14 |
| 2.7 | Parallel fault simulation | 16 |
| 2.8 | Modeling faults in parallel fault simulation | 17 |
| 2.9 | D-Algorithm Example Circuit Diagram | 21 |
| 2.10 | PODEM Example Circuit Diagram | 25 |
| 3.1 | Generic backtrack search algorithm | 35 |
| 3.2 | Logic Circuit | 40 |
| 3.3 | Directed Acyclic Graph | 40 |
| 3.4 | Faulty Circuit with Wire D Stuck-at 1 | 41 |
| 3.5 | XOR of Good and Faulty Circuits | 42 |
| 4.1 | GRASP SAT Solver | 64 |

| | | |
|------|---|-----|
| 4.2 | SATO SAT Solver | 66 |
| 4.3 | Chaff SAT Solver | 67 |
| 4.4 | Chaff SAT Solver Watching Literals Technique | 68 |
| 4.5 | Chaff SAT Solver Conflict Analysis Implication Graph | 71 |
| 4.6 | Example CNF File | 73 |
| 5.1 | SCDJ Flowchart | 80 |
| 5.2 | SCDJ Algorithm | 83 |
| 5.3 | Normalized Number of Decisions | 105 |
| 5.4 | Normalized Number of Implications | 106 |
| 5.5 | Normalized Number of Conflicts | 107 |
| 5.6 | Normalized Runtime | 108 |
| 5.7 | Normalized Number of Decisions with Clause/Variable ratio < 3.0 | 109 |
| 5.8 | Normalized Number of Implications with Clause/Variable ratio < 3.0 | 110 |
| 5.9 | Normalized Number of Conflicts with Clause/Variable ratio < 3.0 | 111 |
| 5.10 | Normalized Runtime with Clause/Variable ratio < 3.0 | 112 |
| 6.1 | Example BLIF File | 118 |
| 6.2 | C17 Test Patterns | 122 |
| 6.3 | C17 Circuit Model | 123 |
| 6.4 | Application-Specific Conflict Diagnosis-Based SAT Solver | 125 |
| 6.5 | Hardware SAT Solver Configuration Algorithm for ATPG | 127 |

List of Tables

| | | |
|-----|---|----|
| 2.1 | Singular Covers for the Circuit in Figure 2.9 | 22 |
| 2.2 | Propagation D-cubes for the Circuit in Figure 2.9 | 22 |
| 2.3 | D-Algorithm Steps of Operation for the Circuit in Figure 2.9 . . . | 23 |
| 2.4 | PODEM Steps of Operation for the Circuit in Figure 2.10 | 26 |
| 3.1 | Basic Gate Formulas | 39 |
| 4.1 | DPLL Execution Example | 61 |
| 5.1 | SCDJ and zChaff Common Behavior for Test Instance “aim-50-1.6-yes1-1” | 85 |
| 5.2 | zChaff Behavior after Conflict Analysis for Test Instance “aim-50-1.6-yes1-1” | 86 |
| 5.3 | SCDJ Behavior after Conflict Analysis for Test Instance “aim-50-1.6-yes1-1” | 87 |
| 5.4 | DIMACS Benchmark Suite Results - part 1 | 90 |
| 5.5 | DIMACS Benchmark Suite Results - part 2 | 91 |
| 5.6 | DIMACS Benchmark Suite Results - part 3 | 92 |
| 5.7 | DIMACS Benchmark Suite Results - part 4 | 93 |

| | | |
|------|--|-----|
| 5.8 | DIMACS Benchmark Suite Results - part 5 | 94 |
| 5.9 | DIMACS Benchmark Suite Results - part 6 | 95 |
| 5.10 | DIMACS Benchmark Suite Results - part 7 | 96 |
| 5.11 | DIMACS Benchmark Suite Results - part 8 | 97 |
| 5.12 | DIMACS Benchmark Suite Results - part 9 | 98 |
| 5.13 | DIMACS Benchmark Suite Results - part 10 | 99 |
| 5.14 | DIMACS Benchmark Suite Results - part 11 | 100 |
| 5.15 | DIMACS Benchmark Suite Results - part 12 | 101 |
| 5.16 | DIMACS Benchmark Suite Results - part 13 | 102 |
| 5.17 | DIMACS Benchmark Suite Results - part 14 | 103 |
| 5.18 | DIMACS Benchmark Suite Results - part 15 | 104 |
| 5.19 | Average normalized values with clause/variable ratio < 3.0 . . . | 113 |
| 6.1 | ISCAS85 Benchmark Results | 121 |

List of abbreviations

| | |
|-------|---|
| ATPG | Automatic Test Pattern Generation |
| BCP | Boolean Constraint Propagation |
| BDD | Binary Decision Diagrams |
| BLIF | Berkeley Logic Interchange Format |
| BRAM | Block Random Access Memory |
| CNF | Conjunctive Normal Form |
| CUT | Circuit Under Test |
| DPLL | Davis, Putnam, Logemann and Loveland search algorithm |
| FDA | Failure-Driven Assertions |
| ISCAS | International Symposium on Circuits and Systems |
| PDF | Primitive D-cube of Failure |
| RPG | Random Pattern Generation |
| RTL | Register Transfer Level |

| | |
|-------|---|
| SAT | Boolean Satisfiability |
| SCDJ | Selective Conflict Directed Jumping |
| UIP | Unique Implication Points |
| VLSI | Very Large Scale Integration |
| VSIDS | Variable State Independent Decaying Sum |