Hierarchical Test Generation for Digital Circuits Represented by Decision Diagrams

Jaan Raik

Tallinn 2001

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Science in Computer Engineering in Tallinn Technical University

Abstract

The evolution of microelectronics systems is following the famous Moore‘s Law, which states that their most important characteristics are improved twice in every 18 months. The ever-increasing complexity of these systems constitutes the need for developing new and more efficient test methods.

In this thesis, a new hierarchical test generation approach basing on multi-level Decision Diagram (DD) design representations is presented. A hierarchical circuit model description of multi-level decision diagrams is proposed. This type of description allows us to use the same formalism and mathematical basis on different design abstraction levels.

We discuss the cycle-based simulation paradigm of register-transfer level circuits and show how decision diagram models can accelerate the simulation process. We introduce advanced cycle-based techniques that significantly speed up the register-transfer level simulation.

In addition, a new hierarchical fault model is proposed achieving 100 % stuck-at coverage for the circuit datapath. A test pattern generation technique based on this model is presented. In this approach, differently from known methods, control part and datapath are handled in a uniform manner.

Experiments show that the proposed test generation technique achieves high fault coverages with a speed that is considerably faster than in previously published methods. However, as at the high-level the technique is relying on a functional fault model, the generated test sets are relatively long.

As a solution, a new static test set compaction method is proposed for minimization of sequential circuit tests that are divided into independent test sequences. The method provides more efficient compaction in a much shorter time than previously published works for solving this task.

Acknowledgements

First of all I would like to thank my supervisor, Prof. Raimund Ubar, for leading me to this exciting and challenging topic. He has given me tremendous support and freedom to do the research. Thanks to his great experience in this field he has always offered me a lot of valuable advice.

In addition, I thank Ahto Buldas from Cybernetica company, who helped me to establish the general structure for this thesis and gave a number of useful hints. I would like to thank my colleague Artur Jutman, who pre-reviewed my work and pointed out some mistakes before they reached my opponents.

Special thanks to Adam Morawiec from Grenoble, whose initiative was to research the high-level decision diagram simulation.

I would like to thank my former colleagues Gert Jervan (currently by Linköping University, Sweden) and Jelena Krupnova. Jelena proposed the basic hierarchical test generation approach and a prototype tool in 1995. Gert brought this tool to the state, where intermediate test reports could be obtained.

In summer 1997, the tool basing on the approach presented in this thesis was initially implemented. This work took place in EAS Dresden of the Fraunhofer Institute, Germany. I thank the director of the institute Prof. Günter Elst and also Prof. Karl-Heinz Diener for the facilities and inspiring environment they provided me during the work.

A lot of acknowledgments to different people must be made in respect to the static compaction algorithm proposed in this thesis. The initiator of this task was Jas Takhar, a researcher from UK's Sheffield University. Initial implementation for combinational circuits was done by Antti Markus (currently by Hansabank). Artur Jutman modified the greedy selection function of the approach in order to make the compaction more efficient. I would also like to thank Prof. Matteo Sonza Reorda and Fulvio Corno from the Polytechnical Institute of Torino, Italy, who helped me to solve a number of technical problems in this matter.

Many thanks to director Margus Kruus, my colleagues Eero Ivask, Peeter Ellervee, Elmet Orasson, Marek Mandre, Priidu Paomets, Marina Brik and the entire staff of the Computer Engineering Department of Tallinn Technical University for their support.

Last but not least, I would like to thank my parents, my wife and children.

Jaan Raik
Tallinn, September 2001

Table of Contents

Abstract..
Acknowledgements..
Table of Contents ....
List of Publications..
List of Abbreviations ....
1 Introduction 1
1.1 Test and Reliability of Electronic Systems 1
1.2 Automatic Test Pattern Generation for Digital Circuits 3
1.3 Main Contributions 4
1.4 Outline of the Thesis 6
2 Multi-Level Decision Diagram Representations 7
2.1 Introduction 7
2.2 Decision Diagrams 8
2.3 Register-Transfer Level Decision Diagram Models 12
2.4 Binary Decision Diagrams 16
2.5 Structurally Synthesized BDD Representations 18
2.6 Multi-Level Decision Diagrams 23
2.7 Conclusions 24
3 Simulation Techniques on Decision Diagrams 25
3.1 Introduction 25
3.2 Cycle-Based Simulation on Decision Diagrams 30
3.3 Improved Cycle-Based Techniques 32
3.3.1 Introduction 33
3.3.2 Event-driven cycle-based simulation algorithm 34
3.3.3 Backtracing simulation algorithm 35
3.3.4 Register-transfer triggered simulation algorithm 35
3.4 Experimental Results 36
3.5 Conclusions 39
4 Fault Modeling Using Multi-Level Decision Diagrams 40
4.1 Introduction 40
4.2 Modeling Stuck-At Faults on Structurally Synthesized BDDs 43
4.3 Parallel Fault Analysis on SSBDD Representations 48
4.4 Hierarchical Fault Model for Multi-Level Decision Diagrams 53
4.5 Conclusions 55

5 Hierarchical Test Generation on Multi-Level Decision Diagrams 56
5.1 Introduction 56
5.2 Concept of High-Level Test Generation Constraints 59
5.3 Hierarchical Test Generation Algorithm 62
5.3.1 Introduction 62
5.3.2 Fault manifestation 64
5.3.3 Fault effect propagation 67
5.3.4 Constraints justification 71
5.4 Test Generation Example 78
5.5 Experimental Results 82
5.6 Conclusions 83
6 Fast Static Compaction of Sequential Circuit Tests 85
6.1 Introduction 85
6.2 Basic Concepts and Model Representation 88
6.3 Compaction Algorithm 91
6.4 Detecting Lower Bounds and Global Optima 93
6.5 Experimental Results 94
6.6 Conclusions 96
7 Thesis Summary 100
7.1 Conclusions 100
7.2 Future Work 102
References 103

List of Publications

Hierarchical Test Generation

1. J. Raik, R.Ubar. Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations. *Journal of Electronic Testing: Theory and Applications, Kluwer Academic Publishers*. Vol. 16, No. 3, pp. 213-226, June, 2000.
2. J.Raik, R.Ubar. Sequential Circuit Test Generation Using Decision Diagram Models, *Proceedings of the DATE Conference*, pp. 736-740, Munich, Germany, March 9-12, 1999.
3. J.Raik, R.Ubar. High-Level Path Activation Technique to Speed Up Sequential Circuit Test Generation, *Proc. of the European Test Workshop*, pp. 84-89, Konstanz, Germany, May 25-28, 1999.
4. R.Ubar, J.Raik. Efficient Hierarchical Approach to Test Generation for Digital Systems, *International Symposium on Quality of Electronic Design*, pp. 189-195, March 20-22, 2000, San Jose, California, USA.
5. M.Brik, G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. Hierarchical Test Generation for Digital Systems. *Mixed Design of Integrated Circuits and Systems*, Kluwer Academic Publishers, pp. 131-136, 1998.
6. M.Brik, G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. Mixed-Level Test Generator for Digital Systems. *Proc. of the Estonian Acad. of Sci. Engng.*, Vol. 3, No. 4, pp. 269-280, Tallinn, Estonia, 1997.
7. G.Jervan, A.Markus, J.Raik, R.Ubar. Hierarchical Test Generation with Multi-Level Decision Diagram Models. *Proc. of the 7-th IEEE North Atlantic Test Workshop*, pp. 26-33, West Greenwich, RI, USA, May 28-29, 1998.
8. R. Ubar, J. Raik. Hierarchical test generation for digital systems based on combining bottom-up and top-down approaches. *Proc. of the World Multiconference SCI'98 / ISAS'98*, pp. 374-381, Orlando, USA, July 12-16, 1998.
9. J.Raik, R.Ubar. High-Level Path Activation Technique to Speed Up Sequential Circuit Test Generation, *Compendium of Papers of the*

European Test Workshop, 5 p., Konstanz, Germany, May 25-28, 1999.

10. G.Jervan, A.Markus, J.Raik, R.Ubar. Assembling Low-Level Tests to High-Level Test Frames. *Proc. of the IEEE 15th NORCHIP Conf.*, pp. 275-280, Tallinn, Estonia, Nov. 10-11, 1997.

11. G.Jervan, P.Eles, Z.Peng, J.Raik, R.Ubar. High-Level Test Synthesis with Hierarchical Test Generation, *Proc. of the NORCHIP Conference*, pp. 291-296, Oslo, Norway, Nov. 8-9, 1999.

12. K.H.Diener, G.Elst, E.Ivask, J.Raik, R.Ubar. FPGA Design Flow with Automated Test Generation, *Proc. of the 11th Workshop on Test Technology and Reliability of Circuits and Systems*, pp. 120-123, Potsdam, Germany, Feb 28-Mar 2, 1999.

13. R.Ubar, J. Raik. Hierarchical Test Generation for Complex Digital Systems with Control and Data Processing Parts, *SEMICON Singapore 1999 Technical Symposium*, pp. 43-52, Singapore, May 3-6 1999.

14. M.Brik, G.Jervan, A.Markus, J.Raik, R.Ubar. A Hierarchical Automatic Test Pattern Generator Based on Using Alternative Graphs. *Proc. of the 4-th International Workshop on Computer Aided Design of Modern Devices and ICs*. pp. 415-420, Poznan, Poland, June 12-14, 1997.

15. G.Jervan, A.Markus, R.Ubar, J.Raik. VHDL based Test Generation System. *Proc. of the 5th Int. Conf. on Electronic Devices and Systems*, pp. 145-148, Brno, Czech Republic, June 11-12, 1998.

16. G.Jervan, A.Markus, R.Ubar, J.Raik. Mixed Level Deterministic - Random Test Generation for Digital Systems. *Proc. of the MIXDES'98 Conf.*, pp. 335-340, Lodz, Poland, June 18-20, 1998.

17. G.Jervan, A.Markus, J.Raik, R.Ubar. DECIDER: A Decision Diagram based Hierarchical Test Generation System. *Proc. of the DDECS'98 Conference*, pp. 269-273, Szczyrk, Poland, September 2-4, 1998.

18. G.Jervan, A.Markus, J.Raik, R.Ubar. A Decision Diagram based Hierarchical Test Generator. *Proc. of the BEC'98 Conference*, pp. 159-162, Tallinn, Estonia, Oct. 7-9, 1998.

19. J.Raik, R.Ubar, G.Jervan, H.Krupnova. A Constraint-Driven Gate-Level Test Generator. *Proc. of the 5-th Baltic Electronics Conference.*

pp. 237-240, Tallinn, Estonia, Oct. 1996.

20. R.Ubar, A.Markus, G.Jervan, J.Raik. Fault Model and Test Synthesis for RISC Processors. *Proc. of the 5-th Baltic Electronics Conference*. pp. 229-232, Tallinn, Estonia, Oct. 1996.
21. J.Raik, P.Paomets. Test Synthesis from Register-Transfer Level Descriptions. *Proc. of the 5-th Baltic Electronics Conference*. pp. 311-314, Tallinn, Estonia, Oct. 1996.

Simulation on High-Level Decision Diagrams

22. R.Ubar, A.Morawiec, J.Raik. Cycle-based Simulation with Decision Diagrams, *Proceedings of the DATE Conference*, pp. 454-458, Munich, Germany, March 9-12, 1999.
23. R.Ubar, A.Morawiec, J.Raik. Back-Tracing and Event-Driven Techniques in High-Level Simulation with Decision Diagrams, *Proc. of the IEEE ISCAS'2000 Conference*, Vol. 1, pp. 208-211, May 28-31, Geneva, Switzerland.
24. A.Morawiec, R.Ubar, J.Raik. Cycle-Based Simulation Algorithms for Digital Systems Using High-Level Decision Diagrams. *Proceedings of the DATE Conference*, p. 743, March 27-30, 2000, Paris, France.
25. R.Ubar, A.Morawiec, J.Raik. High-Level Decision Diagrams for Simulation Performance, *Proc. of the World Multiconference on Systemics, Cybernetics and Informatics*, Vol. IX Industrial Systems, pp. 62-67, July 23-26, 2000, Orlando, Florida, USA.
26. R.Ubar, A.Morawiec, J.Raik. Vector Decision Diagrams for Simulation of Digital Systems. *Proc. of the DDECS'2000 Conference*, pp. 44-51, April 5-7, Smolenice, Slovak Republic.
27. A.Morawiec, J.Raik, R.Ubar. Simulation of Digital Systems with High-Level Decision Diagrams. *Baltic Electronics Conference*, Oct. 8-11, 2000, Tallinn, Estonia.

Test Set Compaction

28. J.Raik, A.Jutman, R.Ubar. Fast Static Compaction of Test Sequences Using Implications and greedy Search. *Digest of European Test Workshop*, Stockholm, May 29 – June 1, 20001, pp.

207-210.

29. J.Raik, A.Jutman, R.Ubar. Fast and Efficient Static Compaction of Test Sequences Based on Greedy Algorithms. *Design and Diagnostics of Electronic Circuits and Systems – DDECS'2001*, Györ, Hungary, April 18-20, 2001, pp.117-122.
30. A.Markus, J.Raik, R.Ubar. Fast and Efficient Static Compaction of Test Sequences Using Bipartite Graph Representation, *Proc. of the Second Electronic Circuits and Systems Conference ECS'99*, pp. 17-20, Bratislava, Slovakia, Sept. 6-8, 1999.
31. J.Raik. Greedy Alternative for the Static Compaction of Sequential Circuit Test Sequences. *Baltic Electronics Conference*, Oct. 8-11, 2000, Tallinn, Estonia.
32. A.Markus, J.Raik, R.Ubar. Test Set Minimization using Bipartite Graphs. *Proc. of the BEC'98 Conference*, pp. 175-178, Tallinn, Estonia, Oct. 7-9, 1998.

Algorithms on Structurally Synthesized BDD Models

33. R.Ubar, J.Raik. Multi-Valued Simulation With Binary Decision Diagrams. *Compedium of Papers of IEEE European Test Workshop*. pp. 28-29, Cagliari, Italy, May 28-30, 1997.
34. J.Raik, R.Ubar. Feasibility of Structurally Synthesized BDD Models for Test Generation. *Compendium of Papers of the European Test Workshop*, pp. 145-146, Barcelona, May 27-29, 1998.
35. E.Ivask, J.Raik, R.Ubar. Fault Oriented Test Pattern Generation for Sequential Circuits Using Genetic Algorithms. *IEEE European Test Workshop*, pp. 319-320, May 23-26, 2000, Cascais, Portugal.
36. J.Raik, R.Ubar. Test Generation with Structurally Synthesized BDD Models. *Proc. of the 5th Int. Conf. on Electronic Devices and Systems*, pp. 66-69, Brno, Czech Republic, June 11-12, 1998.
37. E. Ivask, J.Raik, R.Ubar. Comparison of Genetic and Random Techniques for Test Pattern Generation. *Proc. of the BEC'98 Conference*, pp.163-166, Tallinn, Estonia, Oct. 7-9, 1998.
38. R. Ubar, J. Heinlaid, L. Raun, J. Raik. Calculation of Testability Measures on Structurally Synthesized Binary Decision Diagrams. *Proc. of the BEC'98 Conference*, pp. 179-182, Tallinn, Estonia, Oct.

7-9, 1998.

39. E.Ivask, J.Raik, R.Ubar. Fault-Oriented Test Pattern Generation for Sequential Circuits Using Genetic Algorithms. *Baltic Electronics Conference*, Oct. 8-11, 2000, Tallinn, Estonia.

40. M.Aarna, J.Raik, R.Ubar. Parallel Fault Simulation in Digital Circuits. *Proc. of 42th International Scientific Conference of Riga Technical University*. Riga, October 11-13, 2001. (*To be published*).

Hierarchical Fault Simulation

41. R. Ubar, J. Raik, E. Ivask, M. Brik . Multi-Level Fault Simulation of Digital Systems on Decision Diagrams. *1st International Workshop on Electronic Design, Test & Applications*, Christchurch, New Zealand. (*To be published*).

42. R. Ubar, J. Raik, E. Ivask, M. Brik. Hierarchical Fault Simulation in Digital Systems. *Proceedings of Int. Symp. on Signals, Circuits and Systems SCS'2001*, Iasi, Romania, July 10-11, 2001, pp.181-184.

43. M.Brik, J.Raik, R.Ubar. Hierarchical Fault Simulation for Finite State Machines. *Baltic Electronics Conference*, Oct. 8-11, 2000, Tallinn, Estonia.

Defect-Oriented Testing

44. M.Blyzniuk, T.Cibakova, E.Gramatova, W.Kuzmicz, M.Lobur, W.Pleskacz, J.Raik, R.Ubar. Hierarchical Defect-Oriented Fault Simulation for Digital Circuits. *IEEE European Test Workshop*, pp. 151-156, May 23-26, 2000, Cascais, Portugal.

45. M.Blyzniuk, I.Kazymyra, W.Kuzmicz, W.A.Pleskacz, J.Raik, R.Ubar. Probabilistic Analysis of CMOS Physical Defects in VLSI Circuits for Test Coverage Improvements. *Journal of Microelectronics Reliability*, Elsevier Science Ltd. (To be published).

46. W.Kuzmicz, W.Pleskacz, J.Raik, R.Ubar. Module Level Defect Simulation in Digital Circuits. *Proceedings of the Estonian Academy of Sciences*. (To be published).

47. R.Ubar, W.Kuzmicz, W.Pleskacz, J.Raik. Defect-Oriented Fault Simulation and Test Generation in Digital Circuits. *2nd Int. Symp. on Quality of Electronic Design*, San Jose, California, March 26-28, 2001, pp.365-371.
48. T.Cibakova, M.Fischerova, E.Gramatova, W.Kuzmicz, W.Pleskacz, J.Raik, R.Ubar. Defect-Oriented Test Generation Using Probabilistic Estimation. *MIXDES'01*, Zakopane, Poland, June 21-23, 2001, pp.131-136.
49. T.Cibaková, E.Gramatova, W.Kuzmicz, W.Pleskacz, J.Raik, R.Ubar. Defect-Oriented Library Builder for Functional Test Generation. *Design and Diagnostics of Electronic Circuits and Systems – DDECS'2001*, Györ, Hungary, April 18-20, 2001, pp.163-168.

Fault Tolerant Computing

50. A.Benso, P.Prinetto, M.Rebaudengo, M.Sonza Reorda, J.Raik, R.Ubar. Exploiting High-Level Descriptions for Circuits Fault Tolerance Assessments. *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. pp. 212-216, Paris, France, October 20-22, 1997.

Miscellaneous Publications

51. R.Ubar, A.Buldas, P.Paomets, J.Raik, V.Tulit. A PC-based CAD System for Training Digital Test. *Proc. of the V EUROCHIP Workshop on VLSI Design Training*. pp. 152-157, Dresden, Germany, Oct. 1994.
52. R.Ubar, E.Ivask, P.Paomets, J.Raik. A CAD System for Teaching Digital Test. *Proc. of the 4-th Baltic Conference*. pp. 369-372, Tallinn, Estonia, Oct. 1994.
53. P.Paomets, J.Raik, R.Ubar. A CAD System for ASIC Test and Design. *Exhibition 'Search for Partners' at the Special Session 'European Cooperation in Science, Technology and Education. Workshop on Sampling Theory and Applications*. Riga, Latvia, Sept. 1995.
54. J.Raik, P.Paomets, G.Jervan, A.Markus. Test and Diagnostics Software for Digital Integrated Circuits. Baltic Electronics. #3 1996,

p. 4.

55. R.Ubar, J.Raik, P.Paomets, E.Ivask, G.Jervan, A.Markus. Low-Cost CAD System for Teaching Digital Test. *Proc. of the 1st European Workshop on Microelectronics Education*. p. 48, Villard de Lans, France, Feb. 5-6, 1996.

56. R.Ubar, J.Raik, P.Paomets, E.Ivask, G.Jervan, A.Markus. Low-Cost CAD System for Teaching Digital Test. *Microelectronics Education*. World Scientific Publishing Co. Pte. Ltd. pp. 185-188, Grenoble, France, Feb. 1996.

57. G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. Teaching Test and Design with Turbo Tester Software. *Proc. of the 3rd Advanced Training Course: Mixed Design of Integrated Circuits and Systems MIXDES'96*, pp. 589-594, Lodz, Poland, May 30 - June 1, 1996.

58. G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. CAD Software for Digital Test and Diagnostics. *Proc. of the International Conference on Design and Diagnostics of Electronic Circuits and Systems '97*. pp. 35-40, Beskydy Mountains, Czech Republic, May 12-16, 1997.

59. G.Jervan, A.Markus, J.Raik, R.Ubar. Automatic Test Generation System for VLSI. *Proc. of the 1-st Electronic Circuits and Systems Conference*. pp. 255-258, Bratislava, Slovakia, Sep. 4-5, 1997.

60. G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. A Set of Tools for Estimating Quality of Built-In Self-Test in Digital Circuits. *Proc. of the International Symposium on Signals Circuits and Systems*. pp. 362-365, Iasi, Romania, Oct. 2-3, 1997.

61. G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar. A CAD System for Teaching Digital Test. *Proc. of the 2nd European Workshop on Microelectronics Education*, Kluwer Academic Publishers, pp. 287-290, Noordwijkerhout, the Netherlands, May 14-15, 1998.

62. R.Ubar, E.Orasson, J.Raik, H.-D.Wuttke. Combining Learning, Training and Research in Laboratory Course for Design and Test. *Baltic Electronics Conference*, Oct. 8-11, 2000, Tallinn, Estonia.

63. E.Ivask, R.Ubar, J.Raik, A.Schneider. Internet Based Test Generation and Fault Simulation. *Design and Diagnostics of Electronic Circuits and Systems – DDECS'2001*, Györ, Hungary, April 18-20, 2001, pp.57-60.

List of Abbreviations

AG	Alternative Graph (= Decision Diagram)
ALU	Arithmetical-Logical Unit
ATE	Automated Test Equipment
ATPG	Automated Test Pattern Generation
BDD	Binary Decision Diagram
CAD	Computer-Aided Design
CPU	Central Processing Unit
CUT	Circuit Under Test
DD	Decision Diagram
FFR	Fanout-Free Region
FSM	Finite State Machine
FU	Functional Unit
HDL	Hardware Description Language
MUX	Multiplexer
MUT	Module Under Test
PI	Primary Input
PO	Primary Output
ROBDD	Reduced Ordered Binary Decision Diagram
RODD	Register-Oriented Decision Diagram
RT	Register-Transfer
RTL	Register-Transfer Level
s-a-0	Stuck-At-0
s-a-1	Stuck-At-1
SAG	Structural Alternative Graph
SSA	Single Stuck-At
SSBDD	Structurally Synthesized Binary Decision Diagram
VLSI	Very Large Scale Integrated circuits

1 Introduction

The reliability of electronic systems is no longer merely a topic of limited critical applications in military, aerospace and nuclear industries, where failures may have catastrophic consequences. Nowadays, electronic systems are becoming ubiquitous and their reliability issues are present in all types of consumer applications. In order to guarantee a certain level of reliability, adequate testing of electronic products is required. It is an extremely hard task to test contemporary electronic systems, implementing the current level of knowledge. Furthermore, the ever-increasing complexity of these systems causes the need for developing new, more efficient test methods.

1.1 Test and Reliability of Electronic Systems

Electronic systems consist of hardware and software. In this thesis we consider hardware test only. More specifically, we are targeting digital test, and majority of the hardware in use today is based on digital circuits. The term *testing* does not refer to checking the correctness of the function of the implemented circuit (i.e. *functional verification*). By testing we understand checking for manufacturing correctness. In other words, we check, whether the design implemented in silicon is free of manufacturing defects. Thus, according to their definition, testing and verification are different tasks with different goals.

The quality of test influences the level of reliability of the manufactured electronic circuits in the following way. Let *yield* Y be the fraction of the circuits that are manufactured without defects in a given production process. Knowing the process yield Y and the quality of your test stimuli T, it is possible to calculate the fraction of the bad chips that pass the test. This fraction is called *defect level* DL and it can be found according to the famous formula by Williams and Brown [Will81]:

$$DL = 1 - Y^{1-T}$$

As it can be seen from the formula, there are only two ways to minimize the fraction of defective devices passing the test. One possibility is to maximize the yield Y, the other way is to maximize the test quality T.

Yield is a parameter that is very much dependent of the maturity of the manufacturing process. The newer the process, the lower the yield. As during the recent years, the manufacturing processes are renewed in an increasingly frequent rate, low process yields are going to cause problems.

Test quality T is expressed as the *defect coverage* of the test. However, it is not really practical to measure the exact defect coverage of a test. Mathematical mechanisms, called *fault models*, are needed instead in order to model the actual physical defects. The fraction of modeled faults covered by a test is called *fault coverage*. In practice, fault coverage is used as an approximation of test quality T.

The topic of this thesis is to automatically obtain high-quality tests for digital electronic circuits. This task is called *automated test pattern generation*. The theoretical complexity of test generation is high as the task belongs to the class of NP-complete problems. Already for more than a decade, efficient methods for the subclass of digital circuits, called *combinational circuits*, are known. However, almost all the digital devices produced nowadays belong to the class of sequential circuits, containing feedback loops that make the test generation problem extremely difficult.

In order to be able to test complex electronic circuits, dedicated architectures, called *scannable registers*, are inserted to the circuits to make internal points of the circuit controllable and observable. While, scannable registers help us to solve the test generation problem, they have two major shortcomings. First, at-speed test is not possible in scanned designs. Thus, errors that would occur only at full operating speed may escape the test. Second, making registers of the circuit scannable adds to the silicon area, and therefore, to the cost of the chip. In this thesis we are trying to solve the test generation problem without implementing scan structures.

Test generation can be carried out at different levels of design abstraction. Usually each design abstraction level is represented by different types of models. In this thesis, a hierarchical test pattern generation approach is presented, where differently from known methods, high- and logic-level design information is described by a uniform formalism of decision diagrams.

1.2 Automatic Test Pattern Generation for Digital Circuits

While automatic test pattern generation for combinational digital circuits was considered to be a solved problem already by the end of 1980s, test generation for sequential circuits still remains a major challenge. There have been many different approaches to sequential circuit test proposed over the years. Despite of the diversity of techniques, the achieved fault coverages tend to be unsatisfactory and test generation times long for more complex circuits.

At the gate-level, a number of deterministic test generation algorithms [Nie91, Hsi97] have been implemented. However, the execution times are long and for medium and large circuits mostly unacceptable fault coverages have been achieved. Better performance has been reported of simulation based approaches [Rud94, Cor96], which are generally more robust. However, they are efficient for smaller circuits only and become ineffective when the number of primary inputs and sequential depth of the circuit increase.

Test generation approaches that rely on functional fault models only [Brah84, Gup85, Ward90] do not usually guarantee satisfactory structural level fault coverages. In [FFS98] high fault coverages are achieved by using a behavioral fault model but the main issue of the above approach is still very long execution times.

As a possible solution, hierarchical approaches implementing information from several (usually two) abstraction levels have been proposed. In hierarchical testing, top-down and bottom-up strategies are known. In the bottom-up approach [Mur88, GF00], tests generated at the lower level will be later assembled at the higher abstraction level. Such algorithms ignore the incompleteness problem: constraints imposed by other modules and/or the network structure may prevent test vectors from being assembled.

Top-down approach was introduced to solve this problem by deriving environmental constraints for low-level solutions. Lee and Patel [Lee94] propose such approach for testing microprocessor-like circuits. However, only fault coverages for a set of datapath modules are reported. Overall fault coverage assessments for entire circuits are missing.

Previous works in the area of hierarchical testing have the following main shortcomings:

1. Only the faults in the datapath Functional Units (FU) are targeted.
   This usually results in low fault coverages for the control part as well as for multiplexers, registers and fanout buses of the datapath.

2. Complex symbolic algebra is used in high-level path activation.
   This adds computational overhead to test generation and, in some implementations, it can also cause loss of solutions due to conflicting symbol assignments while activating the high-level symbolic paths.

The aim of the approach proposed in current thesis is to overcome the above mentioned shortcomings. The speed-up of the test generation technique is achieved by implementing simplified fault propagation along single path, and constraint justification, where transparency rules are neglected.

The approach is based on using Decision Diagram (DD) models where, differently from known methods, control unit and datapath are handled in a uniform manner. RT level model of the control and datapath parts is represented by high-level DDs, whereas the gate-level descriptions of RTL blocks are given by Structurally Synthesized Binary Decision Diagrams. The method proposed in the paper combines deterministic and simulation-based techniques for test pattern generation. On the RT-level, deterministic path activating is mixed with simulation-based techniques used in constraints solving. The gate-level local test patterns for components are randomly generated driven by high-level constraints and partial path activation solutions.

1.3 Main Contributions

In this thesis, a test generation approach basing on multi-level Decision Diagram (DD) design representations is presented. In Chapter 2, we introduce the basic concept of general DD models. While the mathematical basis for the special case of DDs, called Binary Decision Diagrams (BDD) [Bry86], has been thoroughly studied, there exists no commonly accepted concept on general DDs. In addition we explain a special class of BDDs, called Structurally Synthesized BDDs proposed in [Uba76]. Finally, the

hierarchical description of multi-level Decision Diagrams is introduced. This description allows us to use the same formalism and mathematical basis on different design abstraction levels.

Simulation is not only an important subtask in hierarchical test pattern generation but is a frequently used routine throughout the entire circuit design cycle. In Chapter 3 we discuss the cycle-based simulation paradigm of register-transfer level circuits and show how Decision Diagram models can accelerate the simulation process. Finally, we introduce advanced cycle-based techniques that significantly speed up the register-transfer level simulation. The work discussed in this Chapter is reported in [UMR99, ISC00]

Selection of the fault model determines the efficiency of test generation and the quality of tests. However, this selection is a question of tradeoff. If a model represents the actual physical defects that occur in circuits too much in detail it will become expensive for computation. On the other hand, a model which is more abstract can be tractable but usually does not guarantee good correspondence with real defects. In this thesis we implement hierarchical fault modeling in order to combine accuracy with tractability.

Chapter 4 presents a new hierarchical fault model. At the low-level the circuit is modeled by Structurally Synthesized BDD (SSBDD) representations. We show that SSBDDs have a number of beneficial properties that make them better suitable for modeling single stuck-at faults than traditional logic level representations. Finally, the hierarchical fault model is proposed, where using a combination of dedicated tests for the terminal and non-terminal nodes, 100 % stuck-at coverage for the circuit datapath is achieved.

In Chapter 5, a novel test pattern generation approach based on multi-level DD representations is presented. The speed-up of the test generation technique is achieved by implementing simplified fault propagation along single path, and constraint justification, where transparency rules are neglected. The approach is based on multi-level DD models where, differently from known methods, control part and datapath are handled in a uniform manner. RT level model of the control and datapath parts is represented by high-level DDs, whereas the gate-level descriptions of RTL blocks are given by Structurally Synthesized Binary Decision Diagrams.

The work presented in Chapter 5 has been reported in [Rai00, Rai99a, Rai99b]

Experiments show that the proposed test generation technique achieves high fault coverages with a speed that is considerably faster than previously published methods. However, as at the high level the technique is relying on a functional fault model, the generated test sets are relatively long. As a solution, in Chapter 6, a new static test set compaction method is proposed for static compaction of sequential circuit tests that are divided into independent test sequences. The method provides more efficient compaction in a much shorter time than previously published works for solving this task.

1.4 Outline of the Thesis

The thesis is organized as follows. In Chapter 2, the basic concept of Decision Diagram (DD) representations is given and representing hierarchical designs by means of multi-level DD models is explained. In Chapter 3, simulation on high-level DD representations is discussed. Chapter 4 considers fault modeling at logic level and multi-level DD models. In Chapter 5, a hierarchical ATPG based on multi-level DD models is presented. Chapter 6 explains how it is possible to further optimize the generated test sets by applying fast static compaction methods. Finally, Chapter 7 provides for the thesis summary with conclusions and the future work.

2 Multi-Level Decision Diagram Representations

Selection of the model representation has a significant impact on the speed, efficiency and accuracy of solving the test generation task. As it will be shown below, decision diagrams provide for an efficient means for both, simulation and diagnosis of discrete systems. In this Chapter the general concept of decision diagram models is given, examples of representing systems on different abstraction levels by special classes of decision diagrams are presented, and finally, describing hierarchical designs by multi-level decision diagrams is explained.

2.1 Introduction

The basic concept of Binary Decision Diagrams (BDD) was introduced by two authors, Ubar and Akers, independently from each other in [Uba76, Ake78], respectively. (In [Uba76] decision diagrams were originally refered to as *alternative graphs*). During the following years a number of works about using decision diagrams for test and simulation purposes were published, including [Uba79, Uba83]. However, it was not until the efficient Boolean manipulation method was presented by Bryant in [Bry86] when this type of representations became widely accepted by the research community.

At present, a large number of the VLSI CAD applications [FFK88, MB88, BCMD90, CMF93], as well as applications from other areas in computer science, are basing on BDDs. Different special classes of BDDs have emerged over the years. These include multi-terminal BDDs [Cla93], edge-valued BDDs [Lai93], binary moment diagrams [Bry95], multi-valued decision diagrams [SKMB90], zero-suppressed BDDs [Min93], functional decision diagrams (FDD) [KSR92], Kronecker FDDs [Dre94] and many others.

In addition to the above mentioned classes of BDDs, in [Uba76, Uba96] the concept of structural alternative graphs or Structurally Synthesized BDDs (SSBDD) was introduced. In Subsection 2.5 we explain more in detail the advantages of this special case of BDDs over the classical model (presented in Subsection 2.4). Our aim is not to present the full classification of BDD classes but rather to show how by using SSBDD

models we can overcome some of the ‘problems’ of BDDs created in the traditional manner using the Shannon’s expansion, and how we benefit from SSBDDs, especially in fault simulation.

In comparison to BDDs which represent Boolean functions, the more general case of Decision Diagrams (DD) representing discrete functions has been a far less studied research area. DDs have been used occasionally in some applications, including RT-level test pattern generation [GF00]. However, there is a lack of commonly accepted terminology. We are trying to fill this gap by defining the concept of decision diagrams in Subsection 2.2.

The main motivation for choosing decision diagrams as the model representation in this thesis lies in the fact that they provide inherent speed-up for simulation and fault analysis. These aspects are explained in Subsection 2.2 and are discussed more in detail in Chapters 3 and 4, respectively. Additional advantage of multi-level decision diagrams is that they allow representation of different design abstraction levels and are therefore especially suitable for hierarchical modeling.

In this thesis we are considering circuits that are represented at two levels: Register-Transfer Level (RTL) and gate level. RTL was selected as the higher hierarchy level because it describes both, high-level function and architectural implementation of the design. If a higher abstraction level had been chosen then information about the actual design implementation would have been missing and thus the quality of the generated tests would have decreased. Representing RTL designs by means of DDs is explained in Subsection 2.3. The concept of multi-level DD model, which is used in this thesis for hierarchical circuit representation is given in Subsection 2.6.

2.2 Decision Diagrams

Decision diagrams are graph representation of discrete functions. A discrete function $y=f(x)$, where $y=(y_1,\dots y_n)$ and $x=(x_1,\dots x_m)$ are vectors is defined on $X=X_1\times\dots\times X_m$ with values $y \in Y = Y_1\times\dots\times Y_n$, and both, the domain X and the range Y are finite sets of values. The values of variables may be Boolean, Boolean vectors, integers.

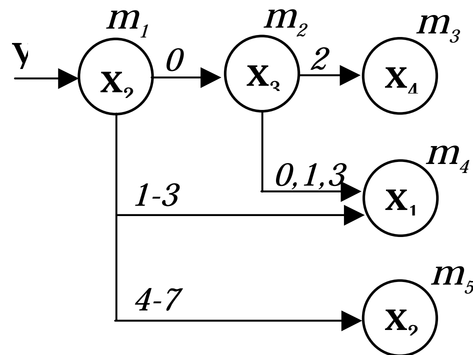


Figure 2.1. Graphical representation of a decision diagram

For representing the functions y=f(x) we use decision diagrams G_y. A Decision Diagram (DD) is a directed non-cyclic labeled graph that can be defined as a quadruple G=(M,E,X,D), where M is a finite set of vertices (referred to as *nodes*), E is a finite set of *edges*, X is a function which defines the *variables labeling the nodes* and the variable domains, and D is a function on E.

The function $X(m_i)$ returns a pair (x_i,X_i), where x_i is the variable letter which is labeling node m_i and X_i is the domain of x_i. Each node of a DD is labeled by a variable. A single variable can label multiple nodes. In special cases of DDs, nodes can be labelled by constants or algebraic expressions.

An edge $e \in E$ of a DD is an ordered pair $e=(m_1,m_2) \in E^2$, where E^2 is the set of all the possible ordered pairs in set E. Graphical interpretation of e is an edge leading from node m_1 to node m_2. It is said that m_1 is a *predecessor node* of m_2, and m_2 is a *successor node* of the node v_1, respectively.

D is a function on E representing the activating conditions of the edges for the simulating procedures. The value of D(e) is a subset of X_i, where $e=(m_i,m_j)$ and $X(m_i)=(x_i,X_i)$. It is required that $Pm_i=\{D(e) \mid e=(m_i,m_j) \in E\}$ is a partition of the set X_i. In other words, the subsets of the set X_i labeled on the edges starting from a node m_i must not overlap and their union must be equal to X_i.

DD has only one starting node (*root node*), for which there are no preceding nodes. The nodes, for which successor nodes are missing are referred to as *terminal nodes*.

Figure 2.1 presents an example of a graphical interpretation of a DD:

$G_y=(M,E,X,D)$,
$M=\{m_1, m_2, m_3, m_4, m_5\}$,
$E=\{e_1, e_2, e_3, e_4, e_5\}$, $e_1=(m_1, m_2)$, $e_2=(m_1, m_4)$, $e_3=(m_1, m_5)$, $e_4=(m_2, m_3)$,
$e_5=(m_2, m_4)$,
$X(m_1)=X(m_5)=(x_2,\{0,1,2,\ldots,7\})$, $X(m_2)=(x_3,\{0,1,2,3\})$, $X(m_3)=(x_4,?)$, $X(m_4)=(x_1,?)$,
$D(e_1)=\{0\}$, $D(e_2)=\{1,2,3\}$, $D(e_3)=\{4,5,6,7\}$, $D(e_4)=\{2\}$, $D(e_5)=\{0,1,3\}$.

Simulation on decision diagrams takes place as follows. Consider a situation, where all the node variables are fixed to some value. According to these values, for each non-terminal node a certain output edge will be chosen which enters into its corresponding successor node. Let us call such connections between nodes *activated edges* under the given values. Succeeding each other, activated edges form in turn *activated paths*. For each combination of values of all the node variables there exists always a corresponding activated path from the root node to some terminal node. Let us call this path the *main activated path*. The simulated value of the variable represented by the DD will be the value of the variable labeling the terminal node of the main activated path.

In Figure 2.2 simulation on the decision diagram presented in Figure 2.1 is shown. Assuming that variable x_2 is equal to 2, a path (marked by bold arrows) is activated from node m_1 (the root node) to a terminal node m_4 labeled by x_1. The value of variable x_1 is 4, thus, $y=x_1=4$. Note that this type of simulation is inherently event-driven since we have to simulate those nodes only (marked by grey color in Figure 2.2) that are traversed by the activated path.

Moreover, DDs are a very efficient model representation for solving diagnostics tasks. For instance, if the single fault model is assumed then locations of faults that can propagate to the output variable of the graph will be restricted to faults in those variables that are labeling the nodes along the main activated path. For example in Figure 2.2 only faults in variables x_1 and x_2 can cause erroneous values of y.

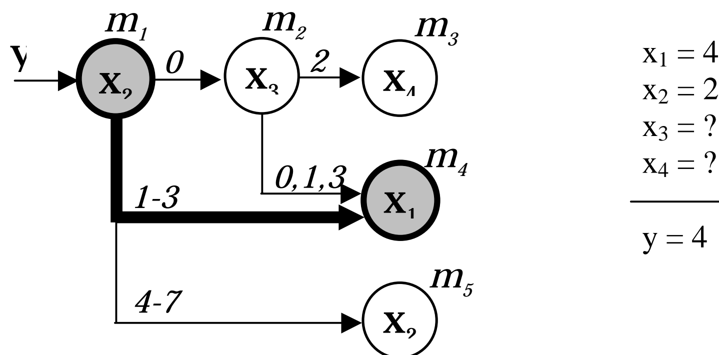


Figure 2.2. Simulation on a decision diagram

In practice, most of the methods in test (for fault simulation and ATPG) and in fault tolerance (e. g. fault injection) fields are based on the single fault assumption. The main motivation for modeling single, as opposed to multiple faults, is the following:

1. The number of fault combinations in the fault list is a serious combinatorial problem. This number rapidly explodes with the growth of the model size.

2. In practice, covering 100 % of single faults in the model is found to cover a vast majority of possible fault combinations.

In addition to the above-mentioned improved simulation and fault analysis, there are other beneficial properties in special classes of decision diagram models. Some of these will be discussed later in this Chapter.

When representing systems and functions by decision diagram models, in general case, a system of DDs rather than a single DD is required. During the simulation in DD systems, the values of some variables labeling the nodes of a decision diagram could be calculated by other decision diagrams of the system.

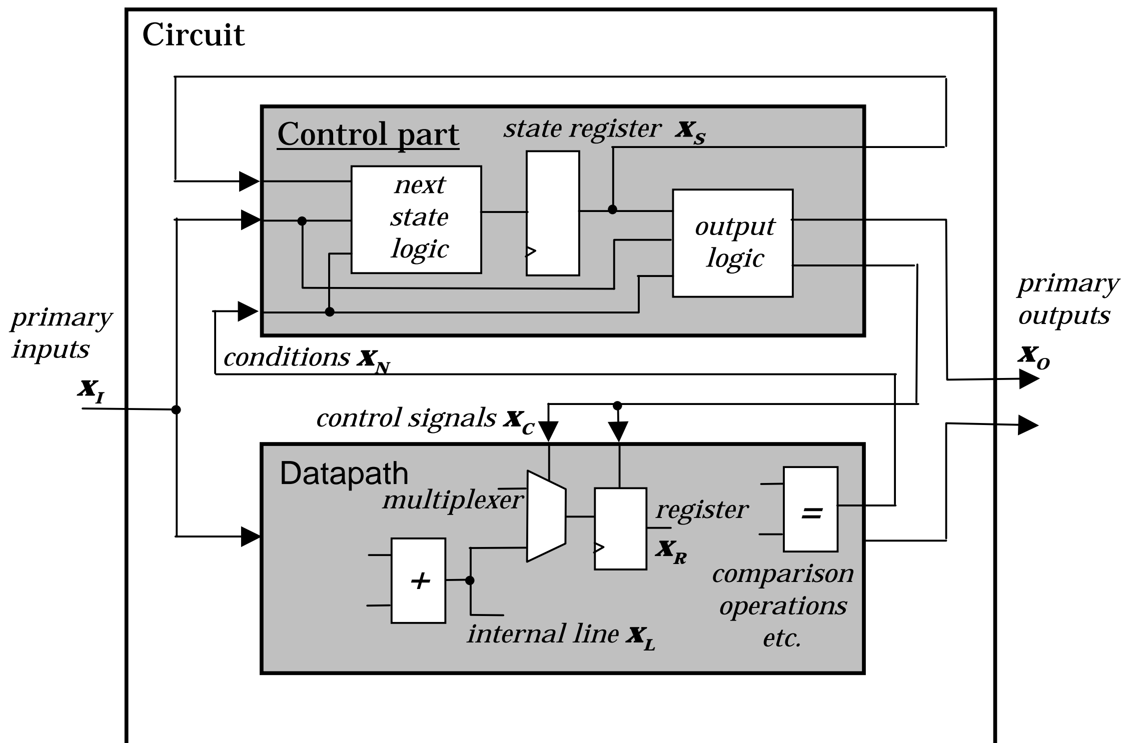
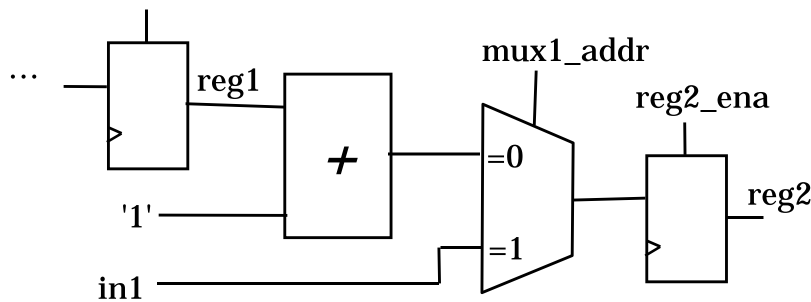


Figure 2.3. Register-transfer level view of a digital circuit

2.3 Register-Transfer Level Decision Diagram Models

As a hierarchical input to the test generator presented in this thesis are descriptions, where the architecture of the circuit is described at the Register-Transfer Level (RTL) and the low-level structure is given at the gate-level. Both of these levels can be described by means of DD models. In this subsection we consider the DD representation at RT-level.

At the RT-level, the design is assumed to be partitioned into a datapath and a control part. Figure 2.3 explains this type of description. Here, the control part is a Finite State Machine (FSM) with a state register (represented by variable x_S in the corresponding DD model), next state logic and output logic. As input signals to the FSM are the primary inputs of the design (variables x_I), conditional signals originating from the datapath (variables x_N) and current value of the state variable x_S. Outputs



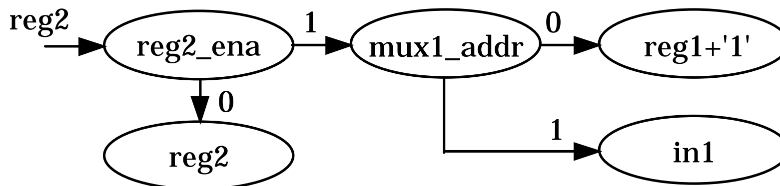


Figure 2.4. A datapath fragment and its DD representation

of the FSM are the primary outputs of the design (variables x_O), control signals (variables x_C) and the next value of x_S.

Datapath can be viewed as a network consisting of modules or blocks. These include registers, multiplexers and functional units (for implementing operations). All the registers and some internal lines of the datapath can be represented by variables in the RTL DD model (variables x_R and x_L, respectively). Inputs for the datapath are the primary inputs x_I and control signals x_C (e.g. multiplexer addresses and register enable signals). Outputs are the primary outputs x_O as well as conditional signals x_N (e.g. from comparison operators) leading to the control part FSM. In the following we will explain how both of these parts can be represented by means of decision diagrams.

In DD models representing the datapath, the non-terminal nodes correspond to control signals (labeled by variables x_C). The terminal nodes

represent operations (functional units). Register transfers and constant assignments are treated as special cases of operations. Figure 2.4 shows a simple example of a DD representation for a datapath fragment.

At the RT-level, datapath is generally represented by a system of DDs. Here, different partitioning strategies are possible. In the hierchical test approach presented in this thesis we use partitioning, where for each primary output, fanout signal and register a DD corresponds. In addition, multiplexers that are connected to an input of an FU are represented by a separate DD. Figure 2.6 shows this type of DD system partitioning for the datapath given in Figure 2.5.

However, it is possible to use alternative partitionings. For example, Figure 2.7 shows an approach, where for each register of the datapath exactly one decision diagram corresponds. We refer to this type of partitioning as Register-Oriented DDs (RODD).

The choice between different types of partitionings is a matter of tradeoff. We selected the former as the model for our test generation algorithm because of the fact that in RODD models a single high-level block can be labeling multiple DD nodes. For example, in Fig. 2.7 the multiplier block is labeling three different nodes. Thus, in this model additional data structures are necessary in order to reflect the relationships between the RTL blocks to be tested and the DD nodes.

On the other hand, the RODD representations can be useful in some applications, e.g. simulation [ISC00]. This will be explained in Chapter 3, where advanced simulation methods on DDs are discussed. Moreover, for the sake of simplicity, the description of the hierarchical test generation algorithm presented in Chapter 5 is given basing on the RODD model.

Similar to the datapath, the control part of an RTL design can be represented by a DD. This DD calculates the values for a vector consisting of the state variable and control signals. In the DD, the non-terminal nodes correspond to current state (labeled by variable x_S) and conditional signals originating from the datapath (variables x_N). Terminal nodes hold vectors with the values of next state and control signals x_C.

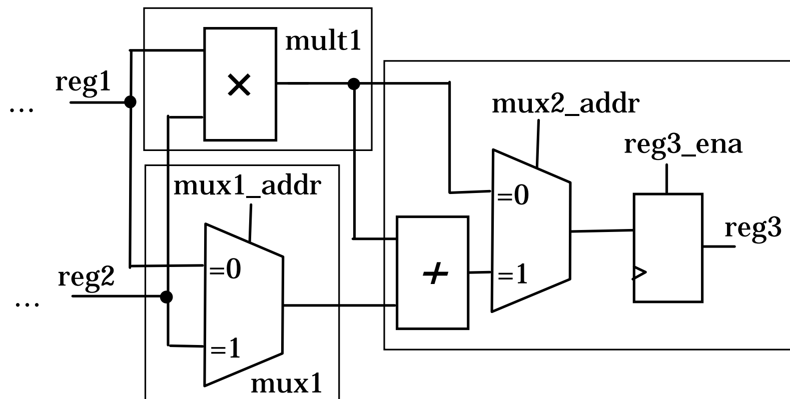


Figure 2.5. Partitioning datapath into DDs

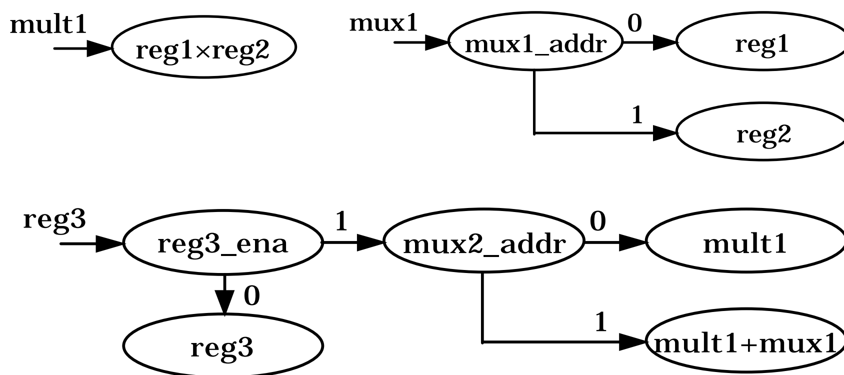


Figure 2.6. System of DDs representing the datapath in Fig. 2.5

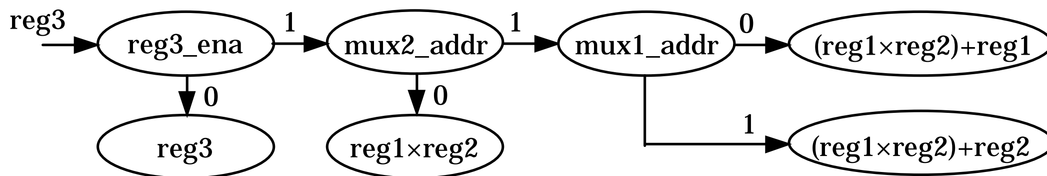
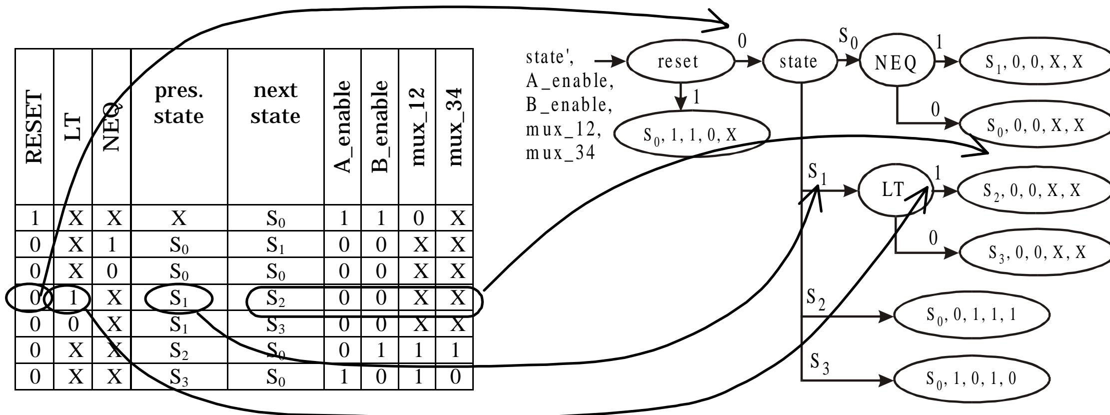


Figure 2.7. RODD representation of the datapath in Fig. 2.5



RESET	LT	NEQ	pres. state	next state	A_enable	B_enable	mux_12	mux_34
1	X	X	X	S_0	1	1	0	X
0	X	1	S_0	S_1	0	0	X	X
0	X	0	S_0	S_0	0	0	X	X
0	1	X	S_1	S_2	0	0	X	X
0	0	X	S_1	S_3	0	0	X	X
0	X	X	S_2	S_0	0	1	1	1
0	X	X	S_3	S_0	1	0	1	0

Figure 2.8. Converting FSM state table into DD

Figure 2.8 shows an FSM state table (representing the control part of the greatest common divisor circuit used as an example later in this thesis) and its corresponding DD representation. In the DD, state' denotes the next state and state denotes the current state value. Variables A_enable, B_enable, mux_12 and mux_34 are FSM outputs and belong to the control signals x_C. Variables RESET, LT and NEQ are FSM inputs and belong to x_N. Figure 2.8 depicts setting up the edges and the terminal node corresponding to the fourth row of the state table.

2.4 Binary Decision Diagrams

This subsection presents the traditional view of BDDs, which are commonly used for representing Boolean functions. We explain the general concept and a special class of Reduced Ordered BDDs (ROBDD) as a canonical form of Boolean functions. However, we do not go through all the special classes and their application areas. In Subsection 2.5 we will introduce another type of BDDs with apparently similar structure but, as we will show, with completely different properties and thus, a different scope of usage.

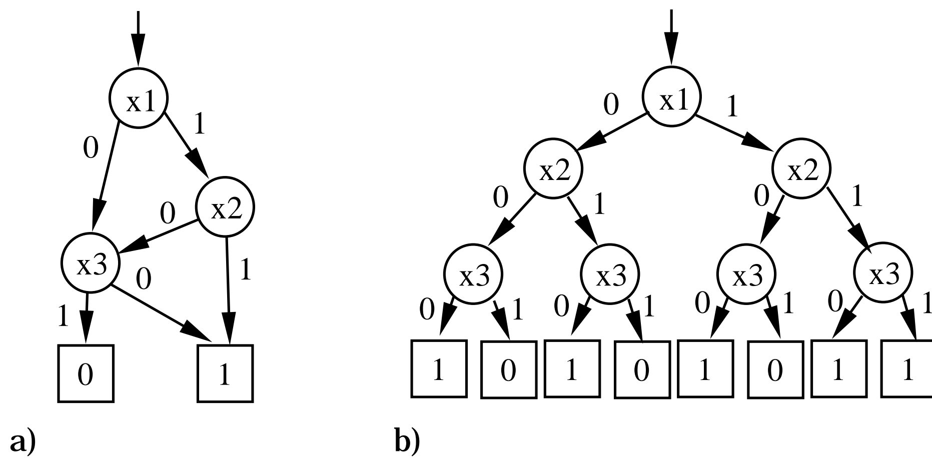


Figure 2.9. A BDD and a binary decision tree for (x1•x2) V $\overline{x3}$

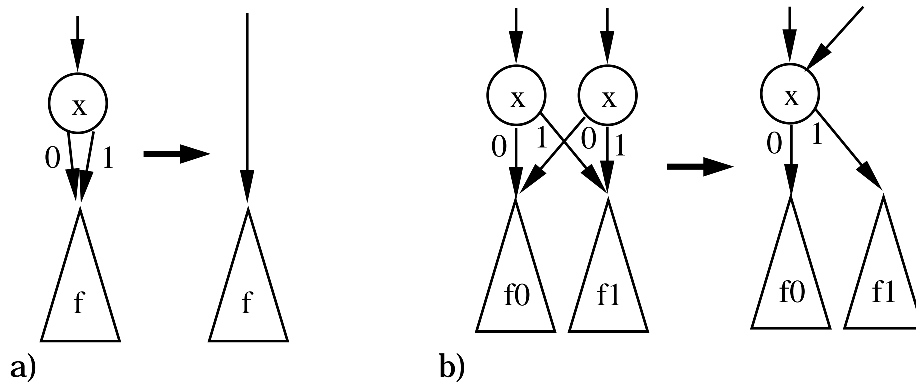


Figure 2.10. Reduction rules for BDDs

A BDD is defined as a directed acyclic graph with two terminal nodes, which we call the *0-terminal* and *1-terminal* node. Each non-terminal node is labeled by an input variable of the Boolean function, and has two outgoing edges, called *0-edge* and 1*-edge*.

An *Ordered BDD* (OBDD) is a BDD, where the input variables appear in a fixed order on all the paths of the graph and no variable appears more than once in a path. Figure 2.9 shows how it is possible to obtain a compact OBDD by reducing a binary tree graph. In the binary tree, 0- and 1-terminal nodes represent logic values 0 and 1, and each node represents

the *Shannon's expansion* of the Boolean function:

$$f = \left(\overline{x_i}.f_0\right) \vee \left(x_i.f_1\right),$$

where i is the index of the variable and f_0 and f_1 are the functions of the nodes pointed to by 0- and 1-edges, respectively.

Reduced Ordered BDD is created by applying the following reduction rules to OBDD.

1. Eliminate all the redundant nodes whose both edges point two the same node (Fig. 2.10a).

2. Share all the equivalent subgraphs (Fig. 2.10b).

The important feature of ROBDDs is that they provide for canonical forms of Boolean functions. This allows us to check the equivalence of two Boolean functions by merely checking isomorphism of their ROBDDs. This is a widely used technique in formal verification.

However, the BDDs as described above have a couple of shortcomings.

1. The total storage space for BDD exceeds 2^n bit of memory in the worst case [LL92]. Thus this type of model representation is impractical in many cases due to the excessive memory requirements.

2. BDDs obtained by Shannon's expansion do not reflect the actual gate-level structure of the circuit. Therefore they are unsuitable to implicitly represent the gate-level structural faults.

In the following subsection we will introduce a different type of BDDs [Uba76], which lack the above mentioned limitations and are very efficient for fault modeling purposes.

2.5 Structurally Synthesized BDD Representations

In current hierarchical test generation approach, gate-level descriptions of the datapath modules are transformed into Structurally Synthesized BDD

(SSBDD) models. SSBDDs were introduced for testing purposes for the first time in [Uba76], i.e. earlier and independent of [Ake78]. In [Uba76], BDDs were originally refered to as functional alternative graphs and SSBDDs as structural alternative graphs, respectively.

The most significant difference between the traditional and SSBDD representations is the method how they are generated. While BDDs are generated by Shannon's (or Davio's [KSR92, Dre94]) expansions, which extracts the function of the logic circuit, the SSBDD models are generated by a superposition procedure that extracts both, function and data about structural paths of the circuit. Another difference between the classical and the SSBDD approach is that in SSBDDs we represent a digital circuit as a system of BDDs, where for each fanout-free region (FFR) of the circuit a separate SSBDD is generated.

Differently from traditional BDDs, SSBDDs support test generation for gate-level structural faults in terms of signal paths without representing these faults explicitly. Furthermore, the worst case complexity and memory requirements for generating SSBDD models for FFRs are linear in respect to the number of logic gates in the circuit, while for traditional BDDs the total storage space exceeds 2^n bits for an n-input combinational circuit [LL92]. Hence, SSBDDs for an arbitrary realistic-sized digital circuit can be generated very rapidly using only a small amount of computer memory.

In Figure 2.11a, an example FFR of a digital circuit is shown. Figure 2.11b gives the BDD of the FFR presented in a *traditional description style*. We can also use an *alternative description style*, where by convention the right-hand edge of a node corresponds to 1 and the lower-hand edge to 0. In addition, terminal nodes holding constants 0, 1 are omitted. Exiting the BDD rightwards corresponds to y =1, and exiting the BDD downwards corresponds to y = 0. Figure 2.11c presents the classical BDD described by using the above-mentioned alternative convention.

SSBDD models for gate-level digital circuits are generated as follows. Starting from the output of the FFR (i.e. primary output or a fanout point of the circuit), logic gates are recursively substituted by their respective elementary BDDs. The procedure of superposition terminates in those nodes, which represent a primary input or a fanout branch of the circuit.

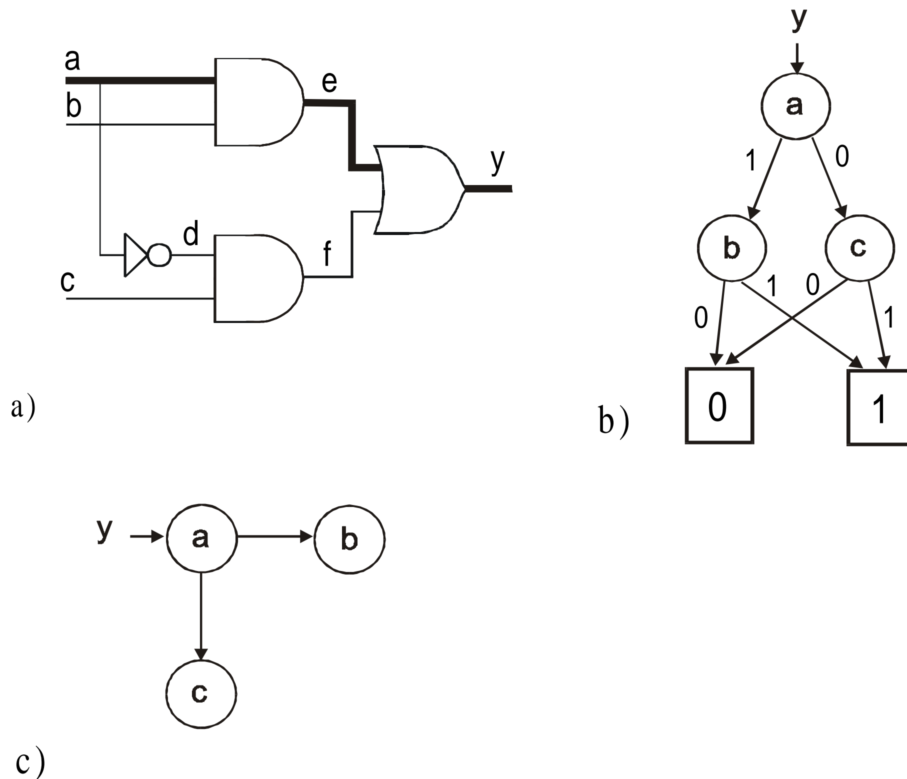


Figure 2.11. BDD representations of a logic circuit

Figure 2.12 shows the superposition-based SSBDD generation for the FFR given in Figure 2.11a using the alternative description style. Note, that in this type of descriptions nodes can be labeled by both, variables x_i and their inversions $\overline{x}_i$. An important property of SSBDD is that each node in it represents a related signal path in the corresponding gate-level circuit. For example, the node denoted by bold circle in Figure 2.12 corresponds to the path marked by bold lines in Figure 2.11a. There are four different signal paths in the circuit and thus, four nodes in the respective SSBDD.

From the above-mentioned property it follows that if we cover the stuck-at faults in all the nodes of the SSBDD (in other words, cover every signal path), we will consequently cover all the stuck-at faults in the corresponding gate-level FFR. Thus, SSBDD models are capable of implicit representation of gate-level stuck-at faults.

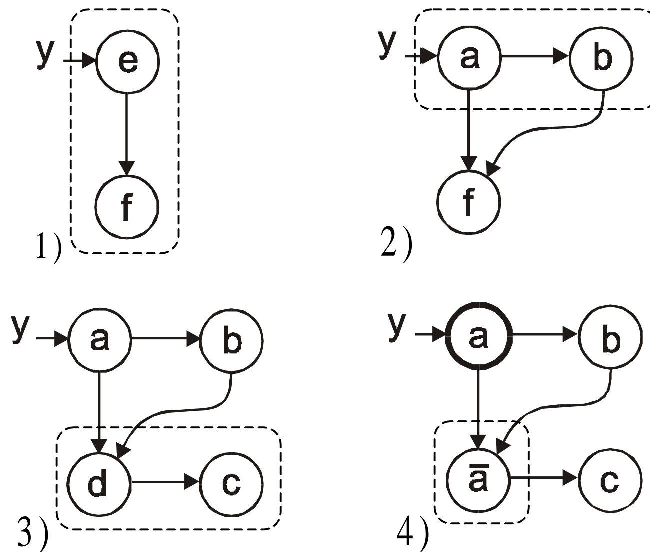


Figure 2.12. Superposition of SSBDD models for Fig. 2.11a

As it was mentioned above, the stuck-at faults in SSBDDs are modeled at nodes and each node corresponds to a distinct path in an FFR. However, the number of signal paths in an FFR is always less or equal than the number of lines between the gates of the FFR. Hence, by generating an SSBDD for a circuit we simultaneously perform collapsing of the faults. However, unlike in traditional fault collapsing, where the aim is simply to minimize the number of faults in the fault list, here, we are at the same time capable of rising to a higher abstraction level of circuit modeling. Fault collapsing using SSBDDs is discussed more in detail in Chapter 4.

While fault simulation [Uba94] and test generation [ETW98] are the main areas of using SSBDD models, this type of representations have been successfully implemented in a wide range of other VLSI CAD applications. These include multi-valued simulation [Uba98], timing simulation [UJP01] and design error diagnosis [Jut00].

Register-transfer level architecture:

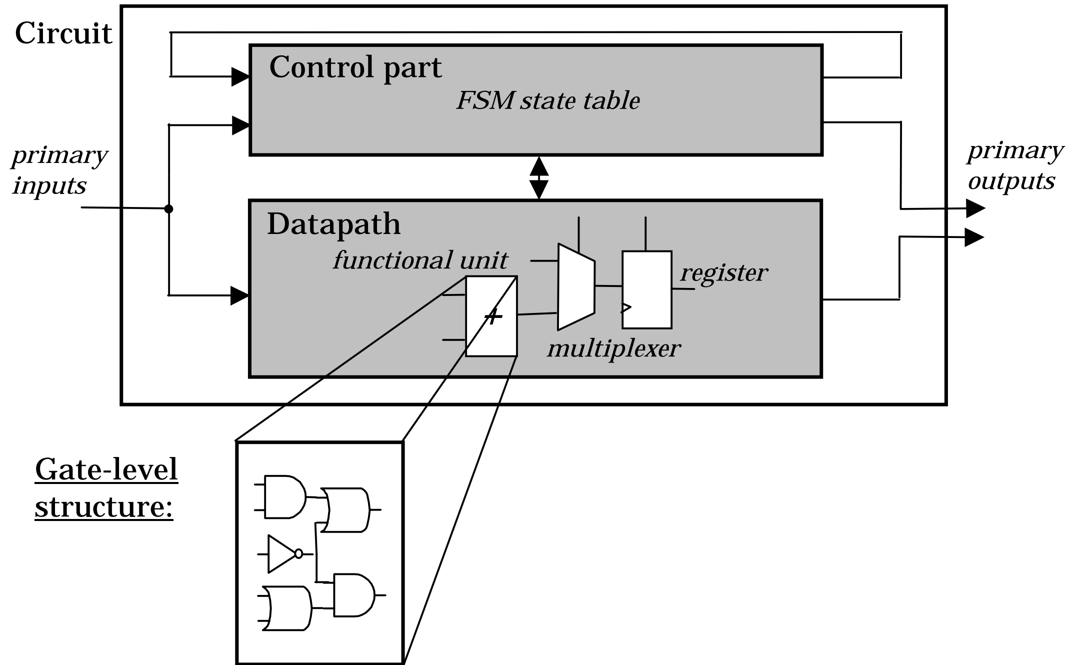


Figure 2.13. Hierarchical description of a digital circuit

Register-transfer level DD model:

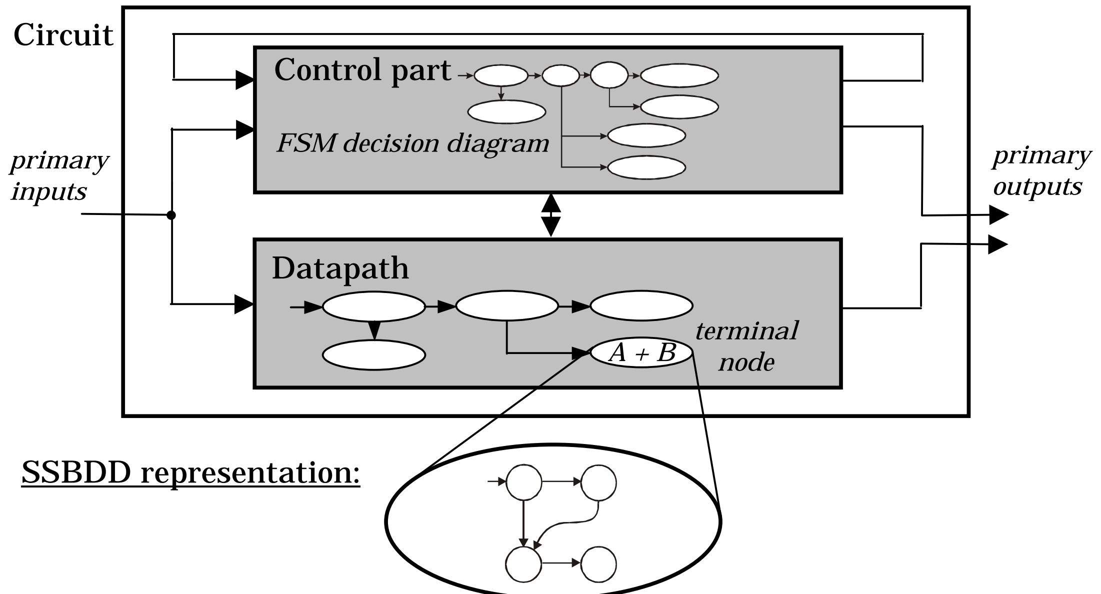


Figure 2.14. Multi-level DD representation of the circuit in Fig. 2.13

2.6 Multi-Level Decision Diagrams

In current thesis, the hierarchical input to the test generator is description, where the architecture of the circuit is described at the Register-Transfer Level (RTL) and the low-level structure is given at the gate-level. Both these levels can be described by means of DD models. As it was mentioned above, at the RT-level we assume that the design is partitioned into a datapath and a control part. The control part is described as an FSM state table and the datapath is presented as a network of modules. The datapath modules include registers, multiplexers and Functional Units (FU).

In the test generation approach presented in this thesis only for the FUs of the datapath the low-level structure is given. This is due to the fact that the RTL fault model introduced in Chapter 4 covers all the low-level structural faults in registers and multiplexers of the datapath, leaving the FUs to be tested at the low-level using the top-down approach (explained in Chapter 5).

As it was mentioned above, the control part is represented in current approach at the high-level. Although only a part of the control part is targeted by the hierarchical fault model proposed in current thesis, as experiments show, a vast majority of structural level faults in the control parts of the example circuits are covered. The hierarchical fault model is discussed in detail in Chapter 4.

Figure 2.12 shows an example of the hierarchical structure of a digital circuit and the gate-level implementation of its FU. Figure 2.13 presents the multi-level decision diagram representation of the circuit. In the multi-level DD representations, the datapath and control part are represented by RTL DD models, while the datapath FUs are represented by SSBDD models.

Note that at the high level the inputs and outputs of FUs are multiple bit buses that are broken down into single bit signals at the low-level. Correspondence between the bit indexes of high-level bus signals and low-level signals must be kept when exchanging values between the two abstraction levels.

2.7 Conclusions

In this Chapter the general concept of decision diagrams was presented. It was shown how using decision diagram representations it is possible to speed up simulation and fault analysis. A special class of binary decision diagrams called Structurally Synthesized BDDs (SSBDD) was explained, which is more suitable for fault modeling purposes than previously presented BDD models. Modeling of RT-level circuits using decision diagrams was presented. Finally, the multi-level DD representations used for hierarchical fault modeling in current thesis were introduced. In the following Chapters we will discuss how to take advantage of the model representations that were defined above.

3 Simulation Techniques on Decision Diagrams

Simulation is not only an important subtask in test pattern generation but is a frequently used routine throughout the entire circuit design cycle. In this Chapter we discuss the cycle-based simulation paradigm of register-transfer level circuits and show how Decision Diagram models can accelerate the simulation process. Finally, we introduce advanced cycle-based techniques that significantly speed up the register-transfer level simulation.

3.1 Introduction

Simulation is an often repeated subroutine in test pattern generation techniques. In test generation, we apply simulation for calculating the responses of the fault-free circuit as well as for evaluating the circuit with the presence of several faults. Thus, introducing faster simulation methods would also accelerate the test generation process as a whole.

In addition to test there are other application areas for simulation. Associated with each design activity is the verification step to be performed in order to ensure the proper functionality of the design. Simulation is a form of design verification, and allows observation of the behavior of the design under specific conditions and input vectors. The complexity of the design to be verified is a critical issue.

There are different approaches to simulation. In Figure 3.1 two alternative strategies for management of the circuit model is shown. Figure 3.1a shows the traditional approach, where the simulation program reads the input stimuli and the circuit model, and calculates the output responses of the stimuli. Better performance can be achieved by compiled-code simulation (see Figure 3.1b), where the circuit model itself is compiled to an executable that reads the vectors, runs simulation and writes the result. However, we are not considering this case in current thesis. The reason for this is that it is very difficul to incorporate such kind of simulator to fulfill the subtasks in a program as complex as hierarchical test pattern generator.

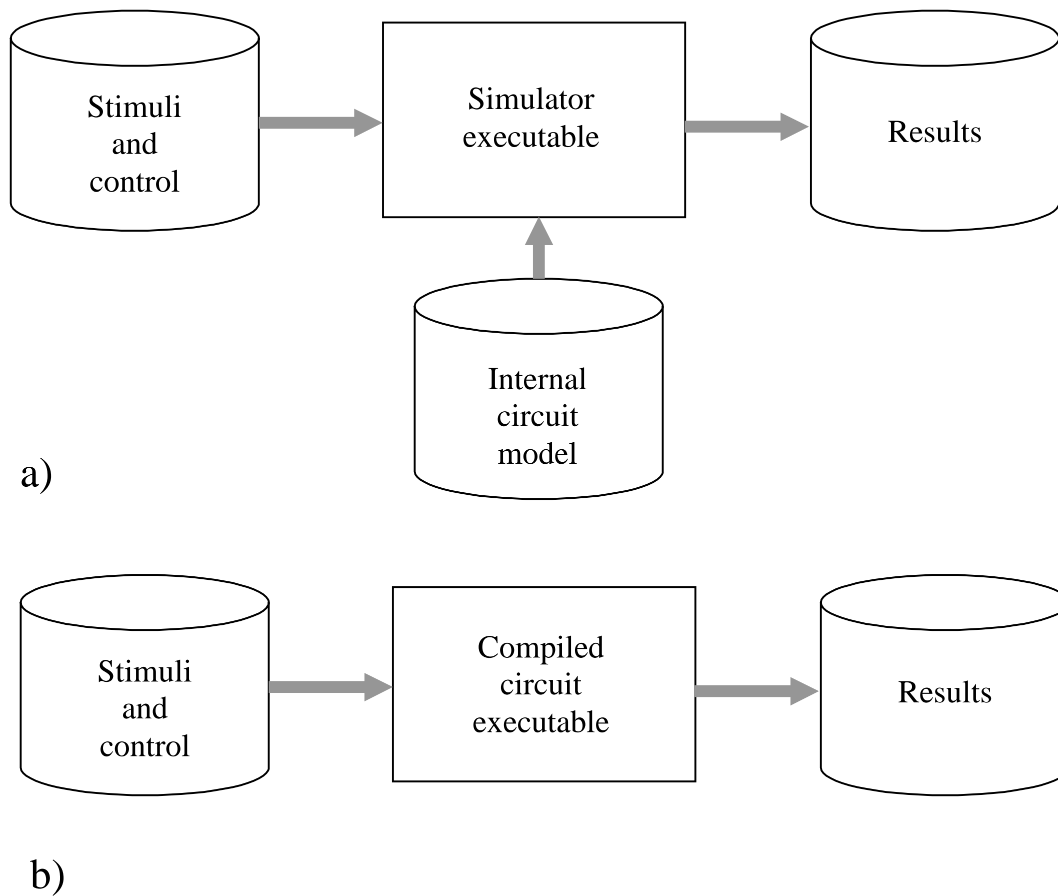


Figure 3.1. Loaded model (a) and compiled model (b) simulation

In the following, when we are refering to simulation we assume RT-level loaded model simulation. The most basic method of this type of simulation calculates the values for all the circuit signals after each elementary time step of the simulation run. The interval between the time step is dependent on the timing resolution of the simulator. Naturally, this approach would be too time consuming for practical applications.

In order to speed up the process, *event-driven* simulation [Meno65, Ulri65] was introduced and is now widely used in VLSI CAD systems. In event-driven simulation signals are evaluated and their values updated only at the presence of signal changes, called *events*. The main operations to be performed in an event-driven simulation algorithm are scheduling of

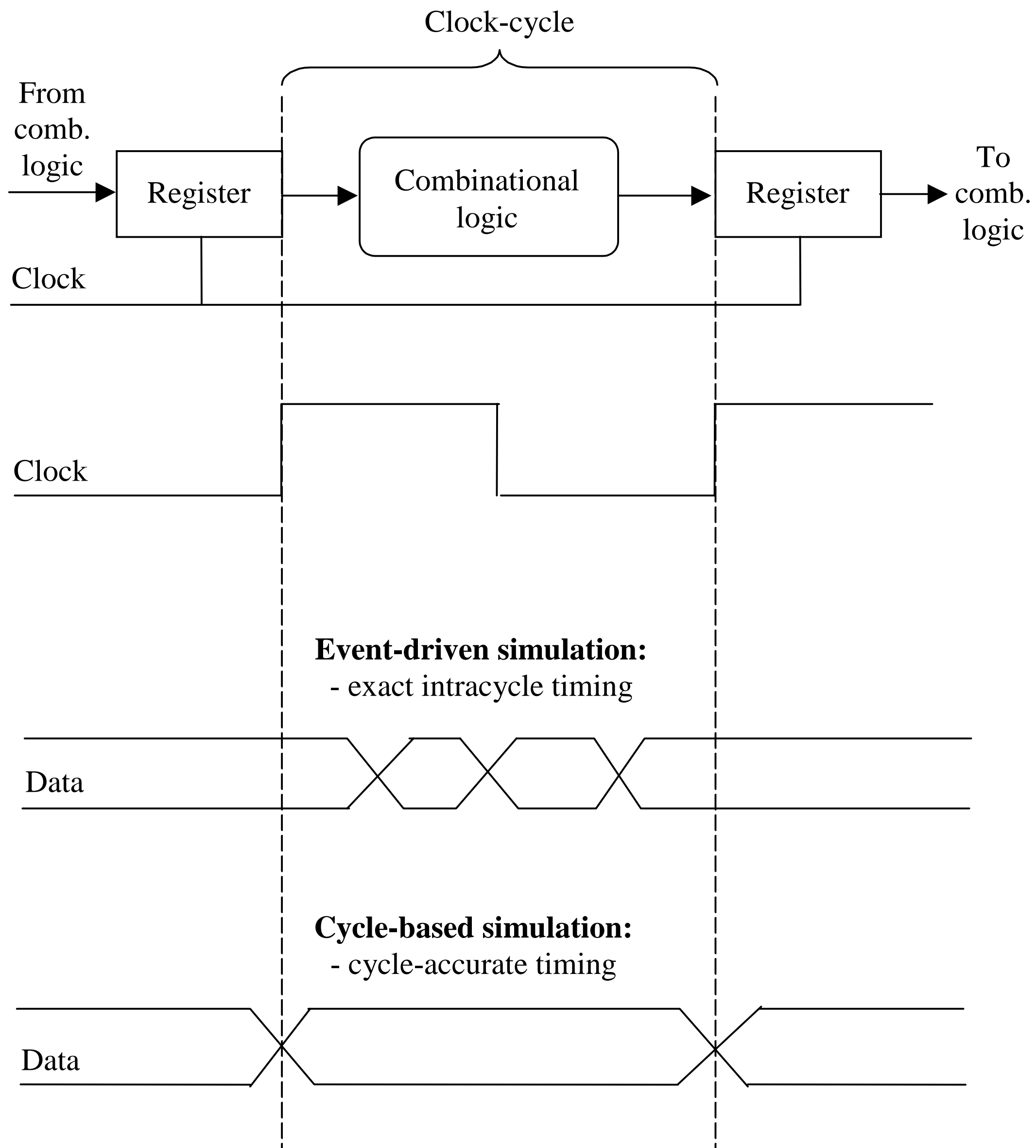


Figure 3.2. Comparison of event-driven and cycle-based simulation

events, processing of events, evaluation and event-list management. All of the above mentioned operations are aplicable to both gate-level and functional level simulation. While event-driven simulation allows detailed results of the behavior of the circuit, it is still computationally expensive.

A vast majority of modern digital circuits are synchronous. *Cycle-based*

simulation is an approach that takes advantage of the synchronizing clock signal of these circuits. It calculates the values for the circuit signals only at the times when the clock front is active, ignoring the intra-clock-cycle timing information. This generally accelerates the simulation process at the expense of resolution and accuracy of timing. In the cases, where circuit timing does not count (e.g. it has been checked or guaranteed) this loss in resolution is fully acceptable. Comparison of cycle-based and event-driven simulation is provided in Figure 3.2.

In a similar way that event-driven simulation was an improvement to the basic approach it is possible to further speed up cycle-based simulation by implementing *event-driven cycle-based* techniques. Here, our resolution is still the clock period but we do not have to calculate the values for all the signals at every cycle. In this Chapter advanced cycle-based techniques using Decision Diagram representations are presented and compared to each other as well as to state-of-the-art cycle-based simulators.

Several methods have been proposed to overcome the problem of the simulation performance. Gruenbacher et. al. [Grue98] proposes abstraction methods for improving the simulation efficiency while raising the level at which the design is represented. Cycle-based simulation is treated as one of possible timing abstraction. Within the last decade Binary Decision Diagrams (BDDs) have become the state-of-the–art data structure in VLSI CAD for representation and manipulation of Boolean Functions [Min96, Luo98]. McGeer et al. [MMS95] applied MDD representation for bit vector discrete function representation and evaluation for logic level cycle-based simulation. Another approach based on BDDs and branching programs is presented by P.Ashar and S.Malik [AM95] for efficient logic function evaluation. As the previous approach it can be applied for the simulation at the logic level only.

However, none of the earlier methods have applied high-level decision diagrams (DDs) as a representation of the design functionality at the higher level of abstraction and focused on the performance issues of the DD simulation algorithms. In current thesis we show that application of decision diagrams for cycle-based simulation offers an important gain in simulation performance in comparison to the HDL-based simulation approach.

This Chapter is organized as follows. In Subsection 3.2 we introduce the

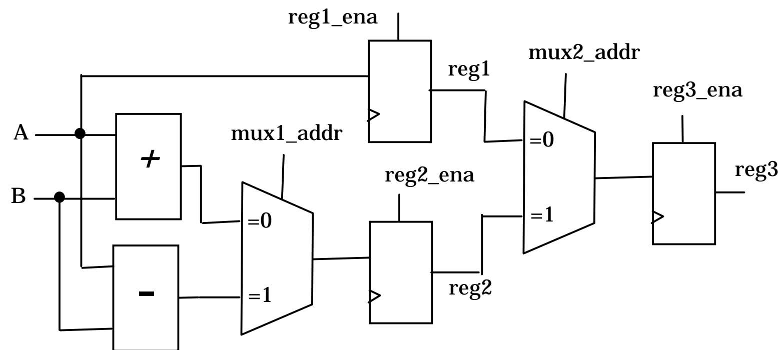
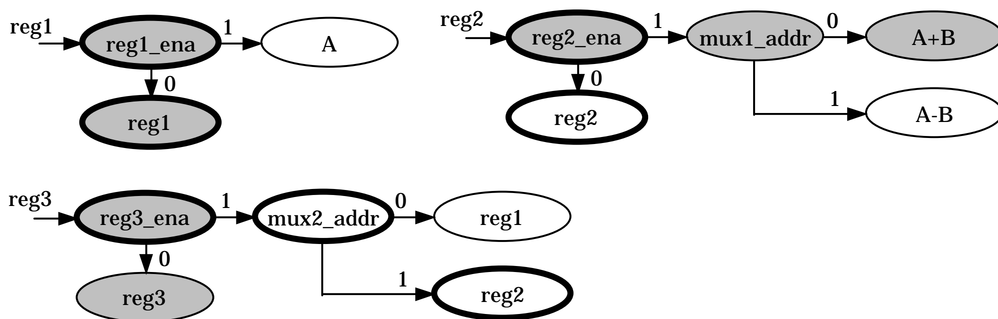


Figure 3.3. Subnetwork of a datapath



	inputs							internal		outp.
	A	B	reg1_ ena	reg2_ ena	reg3_ ena	mux1_ addr	mux2_ addr	reg1	reg2	reg3
vec.1	2	3	0	1	0	0	-	-	5	-
vec.2	**2**	**3**	**0**	**0**	**1**	**0**	**1**	-	**5**	**5**

Figure 3.4. Simulating the DD of the subnetwork in Fig. 3.3

basic concept of cycle-based simulation on Decision Diagram models. Subsection 3.3 presents improved cycle-based techniques on DD representations. In Subsection 3.4 experimental results comparing these techniques to each other and to state-of-the-art simulation tools are given. Finally, conclusions are presented.

3.2 Cycle-Based Simulation on Decision Diagrams

As it was mentioned above, cycle-based simulation of synchronous digital systems is performed on a cycle-by-cycle basis. It assumes that there exist (one or many) clock signals in the circuit and all inputs of the systems remain unchanged while evaluating their values in the simulation (i.e. clock) cycle. The results of simulation report only the final values of the output signals in the current simulation cycle.

The idea of the cycle-based simulation on DDs is the following. The decision diagrams are ranked in such a way that when a DD is simulated its arguments should be all either specified or calculated. Correspondingly, the simulation starts with DDs that depend only on input or state variables, i.e. they are specified either by the input pattern or by the previous state (from the previous clock cycle). Afterwards, also these DDs, which depend on the internal signals that however have been already calculated in the current cycle, are simulated. In such a way there is a need to assess the output value of each DD, but the evaluation is performed only once in a cycle.

Representing register-transfer level circuits by decision diagram models was discussed in Subsection 2.3. For simulation purposes the RODD (Register Oriented Decision Diagram) representation is prefered. Figure 3.3 shows a RODD representation of a datapath subnetwork. In [Lev98] a method was presented for generation of DDs from VHDL, where the fine-grained timing is replaced by a coarse timing. This allows us to get rid of unnecessary details from the model not needed in cycle based-simulation.

As it was mentioned in Chapter 2, simulation on DD models is inherently event-driven, since in DDs during simulation not all the nodes are traced. Only the arguments of traversed nodes are required for evaluation. From that aspect, additional gain in simulation speed is achieved with DDs in comparison to other simulation models.

As an example Figure 3.4 shows simulating the DD model representing the datapath subnetwork depicted in Figure 3.3. Here, simulation of two input vectors (shown in the table in Fig. 3.4) is presented. The nodes traversed during simulation of the first vector are denoted by grey color and those traversed during the second vector are bold. As it can be seen from the figure, only 7 out of 13 nodes are evaluated during the first (as

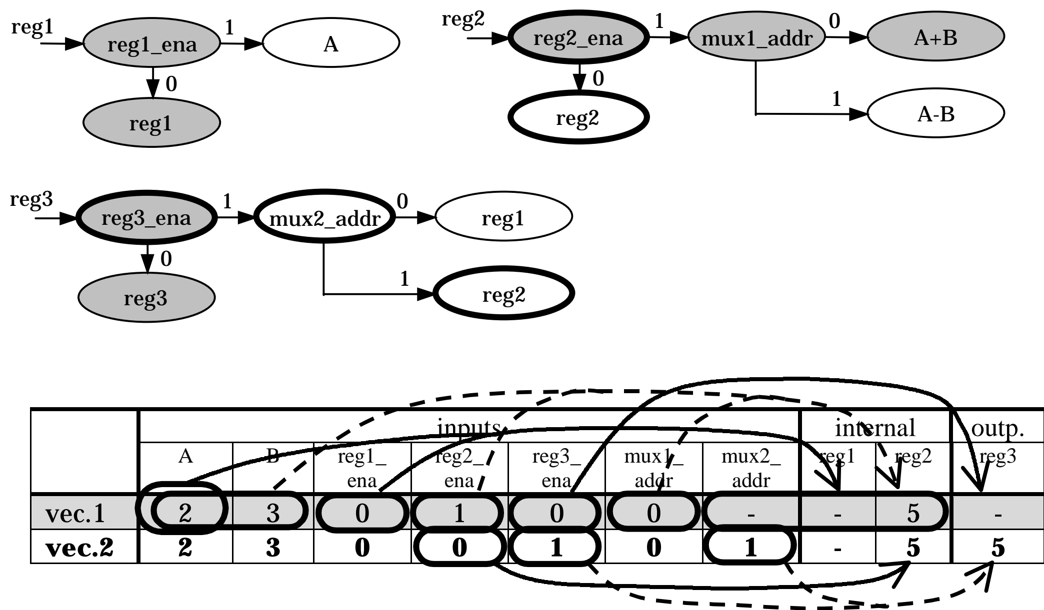


Figure 3.5. Event-driven cycle-based simulation of the subnetwork in Fig. 3.3

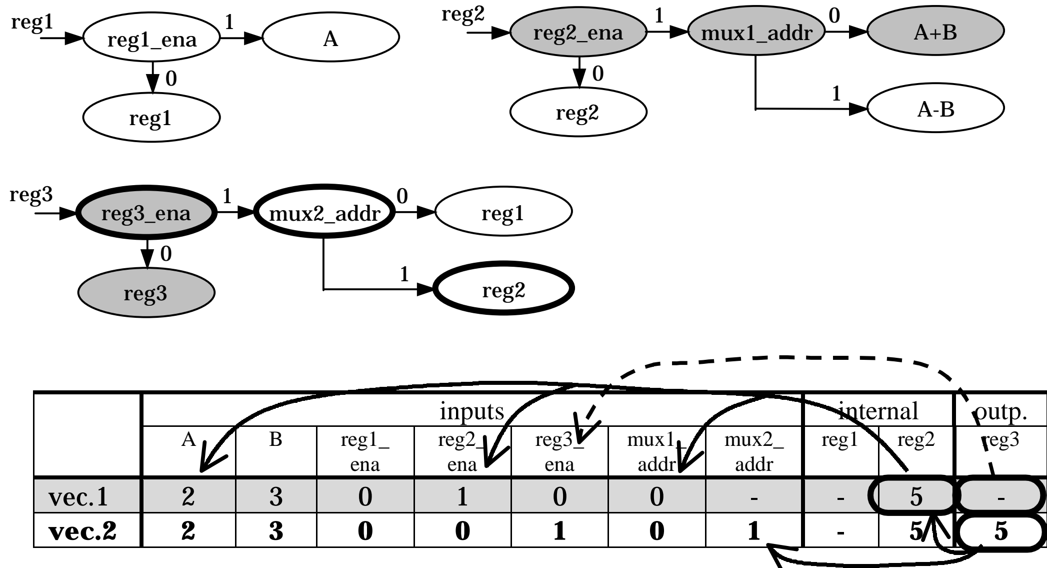
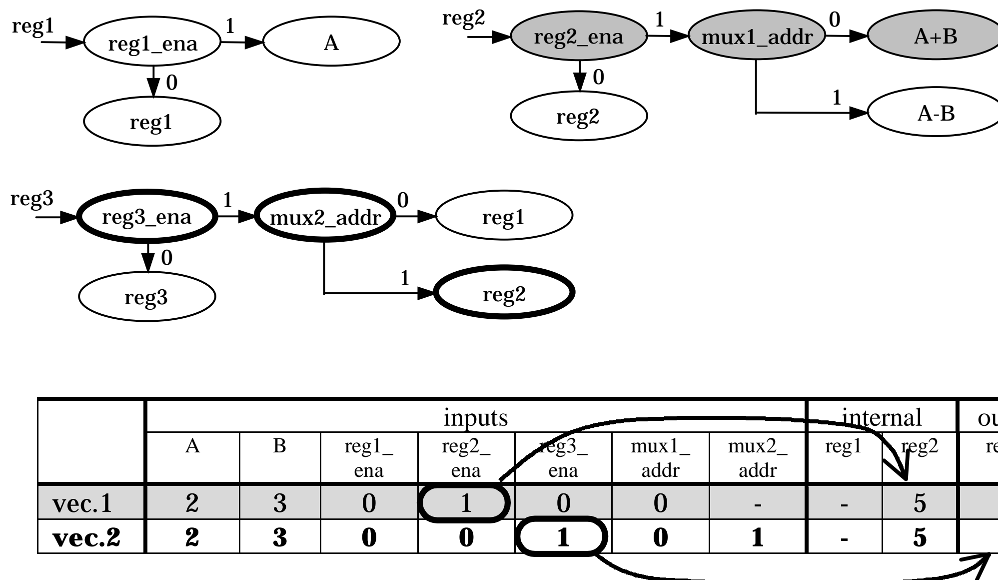


Figure 3.6. Backtracing of the subnetwork in Fig. 3.3



	inputs							internal		outp.
	A	B	reg1_ ena	reg2_ ena	reg3_ ena	mux1_ addr	mux2_ addr	reg1	reg2	reg3
vec.1	2	3	0	1	0	0	-	-	5	-
vec.2	**2**	**3**	**0**	**0**	**1**	**0**	**1**	**-**	**5**	**5**

Figure 3.7. Register-transfer triggered simulation of the subnetwork in Fig. 3.3

well as the second) vector, making it roughly 54 % of the total number of nodes. Thus, the basic cycle-based simulation on DD representations is event-driven and allows us to speed up simulation of RT-level circuits. Traditional cycle-based RTL simulation would have required evaluating all the RT-level blocks of which the design is composed.

3.3 Improved Cycle-Based Techniques

Several approaches can be adapted to handle the evaluation of decision diagrams in cycle-based simulation. The order of evaluation of DDs, observation of the events at their inputs or management of the memory elements in the data path have a critical influence on the performance of the simulation execution. Current Subsection describes four different algorithms proposed for the efficient RODD-model (presented in Subsection 2.3) simulation. The advantages and drawbacks of those algorithms are discussed and compared below.

3.3.1 Introduction

The first approach to simulating the network of interconnected decision diagrams is refered to as 'compute-all' algorithm in this thesis. It evaluates all the diagrams from which the network is composed. This idea was discussed in the previous subsection and it seems to be straightforward and simple to implement (with no additional computational effort necessary).

However, as a drawback of this type of simulation is the fact that it necessitates evaluation of all the diagrams in the network at each simulation cycle even though for some DDs all the inputs, and as a consequence the outputs, do not change in the considered cycle. Thus, for these DDs the produced output variable values remain unchanged. This represents globally an important computational overhead associated with the (useless) evaluation of diagrams, which in case of large systems modeled by a complex network of diagrams and a localized activity in the network can be significant. The minimal number of useful evaluations actually necessary to calculate the new internal or output variable values can be much lower in many practical cases. Subsection 3.3.2 addresses this problem by explaining the *event-driven cycle-based* technique.

On the other hand, as far as circuit's output behavior is concerned, there is no need in simulating those DDs which can not affect the values of primary outputs. This problem, is addressed in Subsection 3.3.3 where a technique based on *recursive backtracing* from primary outputs of the circuit is presented.

Finally, in Subsection 3.3.4 we propose a method refered to as *register-transfer triggered simulation*. Here, only those DDs are simulated that correspond to registers whose enable signals are activated in the present clock cycle. This is a technique especially suitable for the RODD representations and considerably accelerating the cycle-based simulation on decision diagrams.

In other words, there are a number of alternative ways in order to potentially improve cycle-based simulation on DD models. We will discuss a selection of those below.

3.3.2 Event-driven cycle-based simulation algorithm

In order to minimize the number of evaluated decision diagrams in the simulation cycle by avoiding the unnecessary evaluation of those DDs in the network, which do not provide change in the values at their outputs, an event-driven paradigm is adapted for the simulation execution.

In this approach in each simulation cycle the activity of the input variables of every diagram in the network is observed. The evaluation starts from the primary inputs and is propagated towards the primary outputs of the circuit. For every decision diagram the activity of its inputs is checked. If an event (i.e. a change of the value) of at least one of the input variables occurred, the diagram is evaluated. Otherwise, the previous value calculated in the preceding simulation cycle is taken as the actual value for the output variable calculated by the DD.

The above algorithm presents potential improvement in terms of simulation execution time in comparison to the 'compute-all' algorithm, since it allows saving the computation of those DDs, for which the inputs do not change. However, the overall performance is decreased by the additional computational work devoted to keeping track of the input vector events. Another drawback of that algorithm is that it recalculates the new values of DDs even if their outputs do not propagate to the primary outputs of the entire system. This issue is dealt with in Subsection 3.3.3 where the backtracing simulation algorithm is introduced.

Figure 3.5 explains how event-driven algorithm works on decision diagram representations. Here, simulation of two input vectors is shown. The bold rings on the vectors table denote *events*, i.e. value changes of variables assuming that we started off from an all-unspecified state. Events in turn trigger simulation of decision diagrams. This activity is denoted by arrows.

Similarly to Figure 3.4, the nodes traversed during simulation of the first vector are denoted by grey and nodes traversed during the second vector are bold. As it can be seen, the events trigger DD evaluation only five times, while the 'compute-all' algorithm needed six evaluations. 12 out of 26 nodes (2 times 13!) were traversed, which makes 46 % of the total number. (As a comparison, 'compute-all' traversed 54 % in Fig. 3.4).

3.3.3 Backtracing simulation algorithm

An alternative approach focused on the minimization of the number of the DD evaluations is applied in the backtracing algorithm, in which the evaluation of DD is performed according to the propagation of variable values to the outputs of the DD-model.

The backtracing simulation algorithm evaluates the decision diagrams of the network starting from the primary output variables and from the last input vector subsequently moving towards inputs and backwards in time to the first vector. The variable value of a decision diagram G is calculated in a recursive way by evaluating DDs corresponding to variables labeling the nodes traversed by the main activated path of G. In order to avoid multiple evaluation of the same DD during a clock cycle a special flag is used, which shows whether a DD has been simulated. The disadvantage of the above-mentioned approach lies in the fact that the recursive evaluation of DDs is executed even if none of the inputs of the successive DDs changed its value.

Consider Figure 3.6 which shows backtracing simulation for the same DD model and the same vectors as Figures 3.4-3.7. Backtracing starts from the primary outputs at the last vector. In the Figure we backtrace from output reg3 of the second vector to internal register reg2 which poses one clock cycle delay. Thus, at the first vector we have to backtrace both reg3 and reg2. All in all it makes 3 DD evaluations out of 6 and 8 node evaluations out of 26 (i.e. 31 %).

3.3.4 Register-transfer triggered simulation algorithm

An alternative approach to cycle-based simulation of decision diagram representations has been proposed in [ISC00]. The approach is refered to as Register-Transfer Triggered (RTT) simulation algorithm and it is relying on identifying the occurrence of register-transfer operations inside the RT-level network. DDs are evaluated only if during current vector the enable input of the corresponding register allows to write new data to it.

This requires preliminary structural analysis of the RTL circuit in order to extract the information about the register enable signals, which belong to a subset of control signal variables x_C (See Subsection 2.3). However, this analysis has to be carried out only once, prior to simulation.

The main improvement of the proposed RTT algorithm is that it further minimizes the number of evaluated diagrams per simulation cycle according to the activities produced by the control signals. In this way, even if the event occurred at the inputs of a DD corresponding to a register, no evaluation of that diagram is performed unless the register is enabled by the corresponding signal. This is an important advantage over the previous algorithms, which according to the experiments allows to dramatically reduce the time spent for simulating RT-level designs.

The RTT technique is illustrated in Figure 3.7. The preliminary analysis of the DD model identified variables *reg1_ena*, *reg2_ena* and *reg3_ena* as the register enable signals. As a condition that triggers simulation of the respective DD is when its enable signal is '1'. As we can see from the figure, only two DD evaluations are triggered and only 6 node evaluations out of 26 are made. This is two times less than in the case of event-driven technique with the same example.

3.4 Experimental Results

Experiments have been carried out in order to compare the simulation algorithms proposed above. For the experiments four benchmark circuits were used. Gcd is a greatest common divisor circuit [HLS92], mult8x8 is a 8-bit multiplier [FUT95], diffeq is a circuit implementing the differential equation calculation method [HLS92], and huff_enc is a Huffman encoder [DEI99]. Table 3.1 presents the main characteristics and the complexity of the benchmark circuits together with the statistics about their RODD implementation.

One of the ways of comparing the DD simulation algorithms is to compare the number of evaluated decision diagrams for the same set of simulation vectors. In Table 3.2, such comparison of the four considered algorithms is shown. The total number of decision diagram simulations for each of the algorithms as well as the percentage of the diagrams evaluated is given. (For the backtracing algorithm this statistics is not available in the case of the Huffmann encoder circuit because of the size of the circuit and therefore complexity of manual analysis). Minimal results are denoted by bold numbers. During the experiments, real test stimuli generated by the hierarchical test pattern generator presented in current thesis were used.

These test patterns establish actual state sequences in the FSM of the circuit, which allow to emulate realistic circuit behavior, as opposed to random simulation data.

Table 3.1. Characteristics of circuit examples and the corresponding DDs

Circuit	RTL description					RODD description		
	Inputs	Outputs	Registers	FU	MUX	Graphs	Nodes	Variables
gcd	3	1	3	3	2	4	30	13
mult8x8	3	1	7	9	4	8	52	25
diffeq	6	3	7	5	9	8	65	31
huff_enc	5	5	13	22	17	14	202	54

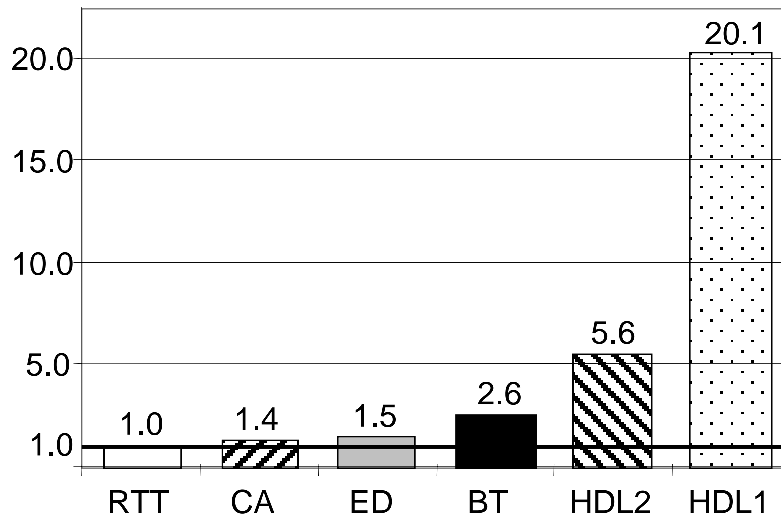
Table 3.2. Comparison of DD simulation algorithms

Circuit	Test length [vectors]	Number of diagram evaluations							
		CA (compute-all)		ED (event-driven)		BT (backtracing)		RTT	
gcd	2949	11792	100 %	8109	69 %	8714	74 %	**4422**	**37 %**
mult8x8	2814	22504	100 %	10894	48 %	15349	68 %	**6385**	**28 %**
diffeq	2081	16640	100 %	10912	66 %	14169	85 %	**6468**	**39 %**
huff_enc	2101	29400	200 %	9302	32 %	N/A	N/A	**5400**	**18 %**

Table 3.3. Experimental results for cycle-based simulation algorithms

Circuit	Simulation time [s] (10 X test length)						Ratio
	CA	ED	BT	RTT	HDL event-driven	HDL cycle-based	HDL cycle / RTT
gcd	0.15	0.19	0.18	**0.11**	1.70	**0.51**	4.64
mult8x8	0.29	0.32	0.35	**0.22**	2.80	**1.00**	4.55
diffeq	0.26	0.33	0.33	**0.21**	6.74	**1.26**	6.00
huff_enc	0.33	0.31	1.10	**0.21**	3.86	**1.40**	6.67

As Table 3.2 shows, in all the cases the Register-Transfer Triggered (RTT) algorithm requires performing the lowest number of diagram evaluations. The difference in terms of evaluated diagrams between the conventional simulation algorithm (referred to as 'compute-all') and the event-driven cycle-based simulation algorithm ranges between 18 % and 38 % for the simulated examples. For more complex models having several levels in the network (e.g. huff_enc) this difference becomes more significant due to the larger number of DDs which were not evaluated.



RTT – Register-transfer triggered simulation on DDs
CA – General cycle-based simulation on DDs ('compute-all')
ED – Event-driven cycle-based simulation on DDs
BT – Cycle-based backtracing on DDs
HDL2 – HDL-based cycle-based simulation
HDL1 – HDL-based event-driven simulation

Figure 3.8. Average normalized run times of simulation algorithms

In addition, Table 3.3 presents the run-times of the benchmark DD-models performed with the use of a DD-based event-driven simulator and the corresponding VHDL models simulation run on state-of-the-art commercial event-driven (VSS, Synopsys) as well as cycle-based (Cyclone, Synopsys) HDL simulation tools. The experiment was run on a 366 MHz SUN UltraSPARC 60 workstation with 512 MB RAM under Solaris 2.5.1 operating system. In order to achieve a better timing resolution all the test sets were multiplied hundred times. For optimal performance, Synopsys tools *cylab* and *cysim* were run with *-perf* and *-2state* options.

Figure 3.8 presents the comparison chart for average run times of the different simulation approaches normalized to that of the RTT algorithm.

3.5 Conclusions

In this Chapter the cycle-based simulation on Decision Diagram models was investigated. Three improvements to the basic cycle-based algorithm were introduced. Subsequent to experiments with these techniques and state-of-the-art commercial tools the following conclusions can be made.

- DDs are an efficient representation for simulation purposes. Basic cycle-based DD simulation offers about 3 – 5 times faster performance than a state-of-the-art cycle-based simulation tool.

- In event-driven and backtracing cycle-based simulation methods the percentage of the DDs evaluated ranges from 32 % up to 85 % in comparison to the conventional approach. However, due to the computational overhead of keeping track of the events, run times are similar or even longer than those of the 'compute-all' approach.

- The Register-Transfer Triggered (RTT) algorithm offers a considerable improvement in comparison to the 'compute-all' approach achieving approximately 25 % to 35 % gain in terms of CPU time.

- The proposed DD-based event-driven simulator (RTT method) is significantly faster than the state-of-the-art cycle-based and event-driven hardware description language (HDL) simulators. The speed-up in simulation time is in average 5.6 times compared to cycle-based HDL simulation and 20.1 times compared to event-driven HDL simulation, respectively.

4 Fault Modeling Using Multi-Level Decision Diagrams

Selection of the fault model determines the efficiency of test generation and the quality of tests. However, this selection is a question of tradeoff. If a model represents the actual physical defects that occur in circuits too much in detail it will become expensive for computation. On the other hand, a model which is more abstract can be tractable but usually does not guarantee good correspondence with real defects. In this Chapter we implement hierarchical fault modeling in order to combine accuracy with tractability. We will provide an overview of existing fault modeling approaches, SSBDD specific fault models and finally, the hierarchical fault model for multi-level decision diagram representations.

4.1 Introduction

The topic of current Chapter is representing physical defects by mathematical abstraction mechanisms called *fault models*. The terms *defect*, *fault* and *error* are repeated throughout this thesis and can be easily confused. However, each of these terms has a different and specific meaning. By *defects* we understand physical failures that occur during manufacturing or because of wear-out in the field. The ultimate goal of test generation is to generate such input stimuli that all (or possibly many) defects would manifest themselves as erroneous output responses of the circuit. In order to be able to describe defects mathematically, an abstraction mechanism called *fault* model is used when generating tests. In general, there is no one-to-one correspondence between faults and defects but the set of faults belonging to the model should represent the defect combinations that are likely to occur in reality. Similar to all the other types of mathematical modeling of physical processes, in choosing the ideal model there is a trade-off between correspondence to real life and tractability of calculations. Finally, the term *error* is related to the behavior of the circuit. Wrong output responses of circuits caused by defects are refered to as errors.

Fault models can be classified according to their level of abstraction into defect-oriented (or transistor level), logic level and high level (register-transfer level and behavioral) ones. At the accurate end of the range are

the transistor level approaches. Stuck-open and resistant bridging faults are the most common fault models at this level. The advantage of these models is that they provide for a more precise modeling of defects. However, calculations are much more complex than on the logic level. On the other hand some of the defect-oriented models (e.g. [UK01]) can be reduced to logic level fault models.

Logic level fault models are by far the ones most commonly used. Different types of delay fault models [Krst98] (e.g. transition faults, path delay faults, gate delay faults), 0-dominant and 1-dominant bridging fault models [Mei74] and the stuck-at fault model belong to this class. The most simple and at the same time the most popular logic level model is the single stuck-at (SSA) fault model, which will be explained in the following Subsection. Its main advantages are stated below.

- It represents a large number of physical defects (see e. g. [Timo83]).

- It is independent of technology. The concept of a signal line being stuck at a logic value can be applied to any structural model.

- Compared to other logic level models, the number of SSA faults is low.

- Many other fault models can be reduced to the SSA model.

In this thesis we will be considering the single stuck-at fault coverage as the measure of test quality. Also the hierarchical fault model of multi-level decision diagrams proposed later in this Chapter is targeting this model. However, this does by no means imply that the presented hierarchical model should be strictly limited to stuck-at faults and cannot be extended to support other logic level models.

As far as sequential circuit test generation is concerned, the main concern with using the stuck-at fault model is tractability. At present there exist no gate-level test generation approach that would guarantee acceptable fault coverages even for moderate sized sequential designs. In order to cope with this complexity alternative fault models on higher levels of design abstraction have emerged.

High-level functional fault models have been proposed for dedicated architectures like microprocessors [Tha80] and finite state machines

[Che90]. Ward and Armstrong presented a fault model for HDL descriptions [Ward90]. However, this model is suitable rather for verification purposes and its defect covering capability is low in most cases. Inability to target real defects is the general shortcoming of functional fault models.

As a solution, hierarchical approaches have been proposed, which take advantage of high level information while generating tests for gate level faults. In hierarchical test generation, *top-down* and *bottom-up* strategies are known. In the bottom-up approach, pre-calculated tests for system components generated on low-level will be assembled at a higher abstraction level [Mur88]. Such algorithms ignore the incompleteness problem: constraints imposed by other modules and/or the network structure may prevent test vectors from being assembled.

Top-down approach [Lee94] was introduced to overcome this problem by deriving global constraints for low-level solutions. The multi-level decision diagram based fault model presented in this thesis is implementing this approach and it has the following advantages over previously proposed models.

1. It targets 100 % of SSA coverage.

2. Control and data parts of the circuit are modeled uniformly (using DDs).

3. It allows us to use the same mathematical basis at different abstraction levels (RTL and logical).

We will continue this Chapter as follows. Subsection 4.2 presents logic level fault modeling on Structurally Synthesized BDD (SSBDD) descriptions, which are the design representations for logic level fault simulation in the hierarchical test generation approach presented in current thesis. Subsection 4.3 explains how SSBDDs can be used for parallel fault analysis, as a further potential to accelerate the SSBDD based fault simulation. Finally, in Subsection 4.4 a new hierarchical fault model based on multi-level decision diagrams is defined.

4.2 Modeling Stuck-At Faults on Structurally Synthesized BDDs

In the stuck-at fault model we assume that the circuit is modeled as an interconnected network of blocks. At logic level these blocks are Boolean gates. A stuck-at fault is assumed to affect only the interconnections between the gates. Each connection can have two types of faults: stuck-at-0 and stuck-at-1 (denoted as s-a-0 and s-a-1, respectively). A line with a stuck-at-1 fault will always hold the logic value 1 irrespective of the correct logic output of the gate driving it.

In general, several stuck-at faults can be simultaneously present in the circuit. A circuit with n lines can have $3^n - 1$ different stuck line combinations. Needless to say that even a moderate value of n will result in an enormous amount of multiple faults. It is a common practice, therefore, to model only single stuck-at faults. An n-line circuit has 2n single stuck-at faults. As we will see later, modeling stuck-at faults on SSBDD representations allows to reduce this number further. Fortunately for the single stuck-at (SSA) model, tests covering all the single stuck-at faults will cover also most of the possible fault combinations.

Single stuck-at fault is characterized by three properties:

1. Only one line is faulty.

2. The faulty line is permanently stuck to 0 or 1.

3. The fault can be at an input or output of a gate.

As it was mentioned above, it is possible to minimize the number of faults to be modeled by a technique called *fault collapsing*. Traditionally, this is done by implementing two types of relations on the set of faults: *fault equivalence* and *fault dominance*. Faults f_1 and f_2 are said to be equivalent if any test that detects f_1 detects f_2 and vice versa, any test detecting f_2 covers also the fault f_1. It is said that fault f_1 dominates f_2, if any test that detects f_2 will also detect f_1. Note that the equivalence relationship is symmetrical while the dominance is not. Equivalence relations between stuck-at faults for basic Boolean gates are presented in Figure 4.1 and the dominance is explained in Figure 4.2.

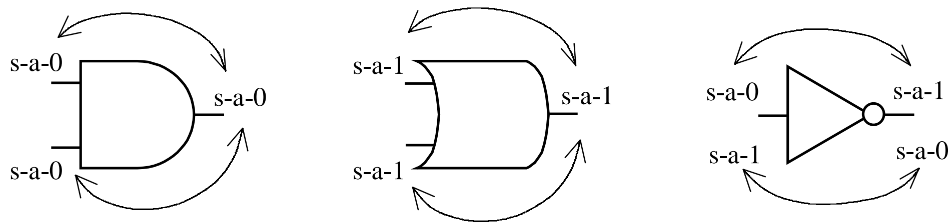


Figure 4.1. Equivalence relationships of stuck-at faults for basic logic gates

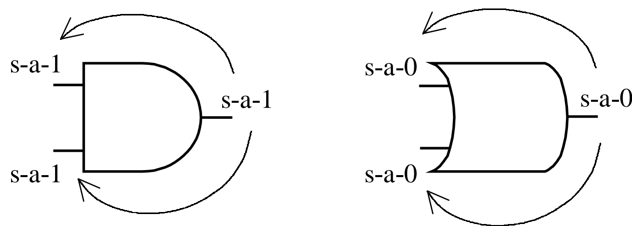


Figure 4.2. Dominance relationships of stuck-at faults for basic logic gates

Consider the circuit in Figure 4.3. The Circuit has 9 lines and thus 18 stuck-at faults (2 faults per each line) initially. However, if we apply the equivalence and dominance operations on the logic gates then only 8 faults (shown in the Figure) will be left for the fault analysis. Table 4.1 presents the collapsed fault list (the left column) and the list of faults dominating these faults (the right column).

Table 4.1. Faults dominating the collapsed faults of the circuit in Fig.4.3.

collapsed fault	dominating faults
a s-a-1	f s-a-0, h s-a-0
b s-a-0	h s-a-0
c s-a-1	g s-a-1
d s-a-1	g s-a-1
e s-a-1	y s-a-1
g s-a-0	c s-a-0, d s-a-0, h s-a-0
h s-a-1	a s-a-0, b s-a-1, e s-a-1, f s-a-1, g s-a-1
y s-a-0	h s-a-0, e s-a-0

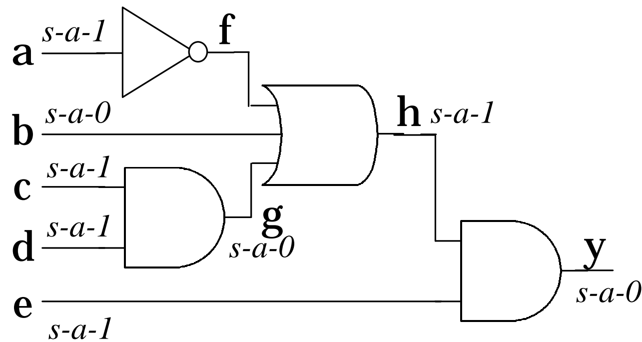


Figure 4.3. Logic level circuit with collapsed fault locations

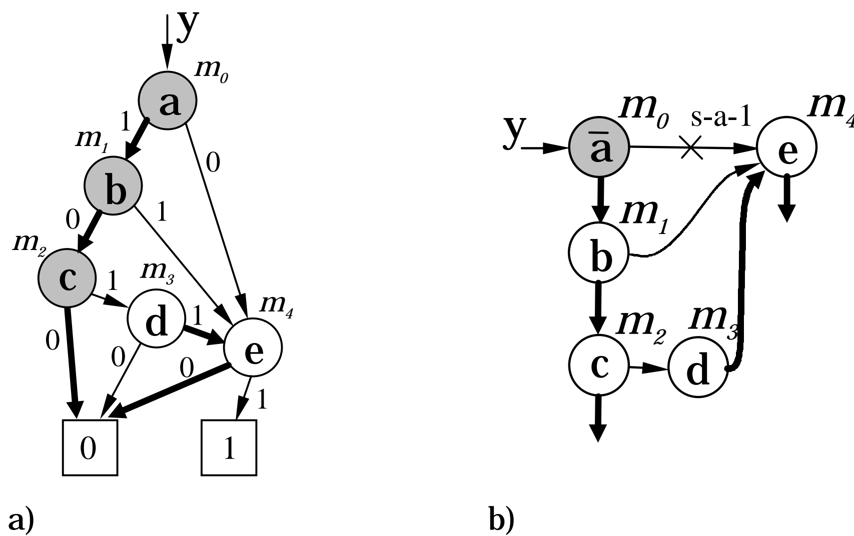


Figure 4.4. Traditional and alternative style SSBDDs for the circuit in Fig. 4.3

Let us discuss now how will it be possible to apply the SSA model to SSBDD representations. Figure 4.4a shows the SSBDD for the circuit in Figure 4.3 presented in a traditional BDD style. Figure 4.4b gives the alternative style description as proposed in [Uba76] of the same SSBDD. In this alternative description the 0- and 1-terminal nodes have been omitted, 0- and 1- edges have been replaced by downward and rightward edges, and nodes can be labeled by both, variables x_i and their inversions $\overline{x}_i$. (See subsection 2.5 for more detailed explanation of traditional and alternative description styles).

While in the gate-level descriptions we model stuck-at faults at the interconnections between the gates, in SSBDD representations the faults are present at nodes. For example, stuck-at-0 fault at a node is modeled with the 0-edge of the node being constantly activated, regardless of the value of the variable labeling this node. As it was explained in Subsection 2.5, each SSBDD node represents a distinct path in the corresponding fanout-free circuit. By testing all the SSBDD node faults we will consequently test all the paths in the circuit and thus all the single stuck-at faults. This ability of SSBDDs to implicitly model logic level stuck-at faults is a very important property, which distinguishes it from other classes of BDDs.

Representing stuck-at faults by faults at SSBDD nodes can be viewed as a type of fault collapsing by applying fault dominance relations along the signal paths of the circuit. For example the uncollapsed fault list of the circuit in Figure 4.3 contains 18 faults. Its corresponding SSBDD has 5 nodes and thus, 10 stuck-at faults. Table 4.2 presents the list of faults in this SSBDD model and the gate-level faults dominating these.

Table 4.2. Faults dominating the SSBDD node faults of circuit in Fig.4.3.

node fault	dominating faults
a s-a-0	f s-a-1, h s-a-1, y s-a-1
a s-a-1	f s-a-0, h s-a-0, y s-a-0
b s-a-0	h s-a-0, y s-a-0
b s-a-1	h s-a-1, y s-a-1
c s-a-0	g s-a-0, h s-a-0, y s-a-0
c s-a-1	g s-a-1, h s-a-1, y s-a-1
d s-a-0	g s-a-0, h s-a-0, y s-a-0
d s-a-1	g s-a-1, h s-a-1, y s-a-1
e s-a-0	y s-a-0
e s-a-1	y s-a-1

Table 4.3 presents the number of uncollapsed faults, collapsed faults and SSBDD faults in eight ISCAS85 circuits. (Two circuits: c432 and c499, were removed from the comparison since they contain exclusive-or gates). As we can see from the Table, the traditional fault collapsing and SSBDD representations provide almost identical results. The difference in the number of faults is at most 8 % (in the case of c1908). SSBDD achieves in

average even 2 % better compaction of the fault list than the traditional approach, reducing the fault lists in average about 1.5 times. However, this advantage occurs partly because in ISCAS models, for every signal line, at least one fault is required to be included to the fault list. In a more advanced collapsing techniques the list can be further minimized. Nevertheless, Table 4.3 indicates that by generating SSBDDs, the fault list will simultaneously be reduced by a considerable amount of faults.

The advantage of the SSBDD based collapsing over the traditional one is that it allows us at the same time to rise to a higher abstraction level of circuit modeling. In the traditional case we would only minimize the number of faults but would still be working at the level of logic gates.

Table 4.3. Number of collapsed and SSBDD faults in ISCAS85 circuits

circuit	uncollapsed	collapsed	SSBDD
c880	1550	**942**	994
c1355	2194	**1574**	1618
c1908	2788	1879	**1732**
c2670	4150	2747	**2626**
c3540	5568	3428	**3296**
c5315	8638	**5350**	5424
c6288	9728	**7744**	**7744**
c7552	11590	7550	**7104**

Finally, we will discuss a couple of additional beneficial properties of the SSBDD models in fault analysis. Consider the circuit in Figure 4.3 and its SSBDD representations in Figure 4.4. Let us assume that we are simulating single stuck-at faults in this circuit with an input vector a=1, b=0, c=0, d=1 and e=0. By these values we will obtain an activated path (denoted by bold arrows) through the SSBDD traversing the nodes m_0, m_1 and m_2. Since the SSBDD represents the behavior of the circuit and no nodes that are outside of the activated path can affect this behavior, no stuck faults in those nodes will be covered. Hence we can omit analysis of faults in these nodes, which in the case of our example means that we have to simulate three faults only (in the nodes denoted by grey in Figure 4.4a).

Let us once again consider this same example and assume that we are simulating the stuck-at-1 fault at node m_0 in Figure 4.4b. Caused by the stuck-at fault the rightward edge of this node is constantly activated, which leads us to node m_4. From this node we exit the graph downwards, which in the case of this type of SSBDD descriptions, means that the value of the decision diagram is zero. Hence the stuck-at-1 fault in node m_0 will not be detected. Since we attempted to move rightwards from node m_0 but ended up in zero value, we can omit all the nodes with indexes between the faulty node (i.e. m_0) and its rightward successor (i.e. m_4) from further analysis. (Please note that the node indexes in SSBDD representations are always ranked this way that a node with a lower index is a predecessor to the node with a higher index). Due to this property, in our example the stuck-at 1 fault in node m_0 is the only fault that we have to analyze with this input vector. This is a tremendous gain in comparison to traditional approach.

Note that the node omission property is not applicable in the traditional style SSBDD descriptions as given in Figure 4.4a. This property of SSBDDs has been succesfully implemented to speed-up fault simulation in [Uba94] and multi-valued simulation in [Uba98].

4.3 Parallel Fault Analysis on SSBDD Representations

Parallel fault simulation takes advantage of the word-oriented nature of operations in the computer to simulate the effects of a number of faults simultaneously [Sesh65]. The number of faults that can be simulated during a simulation pass is determined by the word length n of the host CPU. If the number of faults is large then multiple simulation passes are required.

At first we will consider the simplest case, where two logic values and single stuck-at faults are considered at the gate-level. Here, the logic operation performed by a gate is simulated by performing the corresponding operation on the computer words representing its input values. Figure 4.5 presents simulation of two faults (c stuck-at 0 and f stuck-at 1) in a logic circuit on a computer whose word length is 3 bits. As you can see from the Figure, the fault c stuck-at 0 is detected at the circuit output since its bit value in the corresponding word differs from the first (i.e. fault-free) bit.

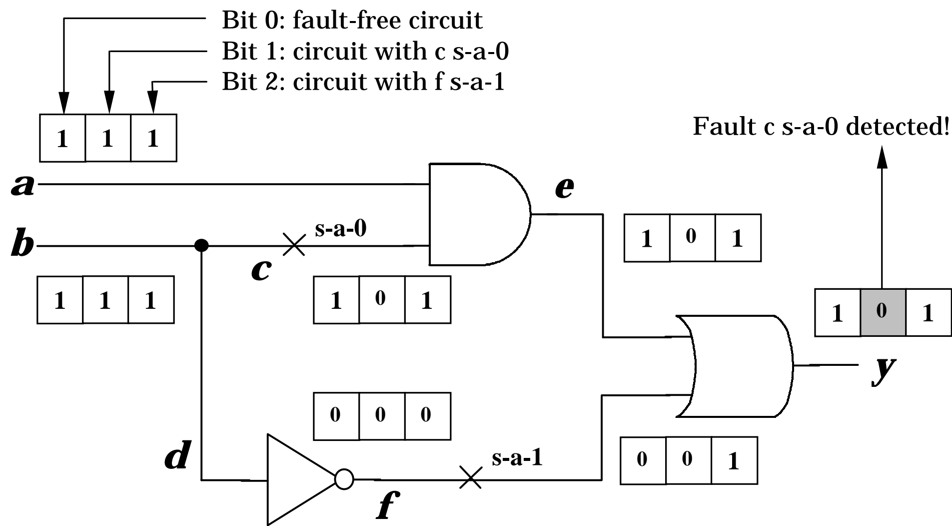


Figure 4.5. Parallel fault simulation on a logic circuit

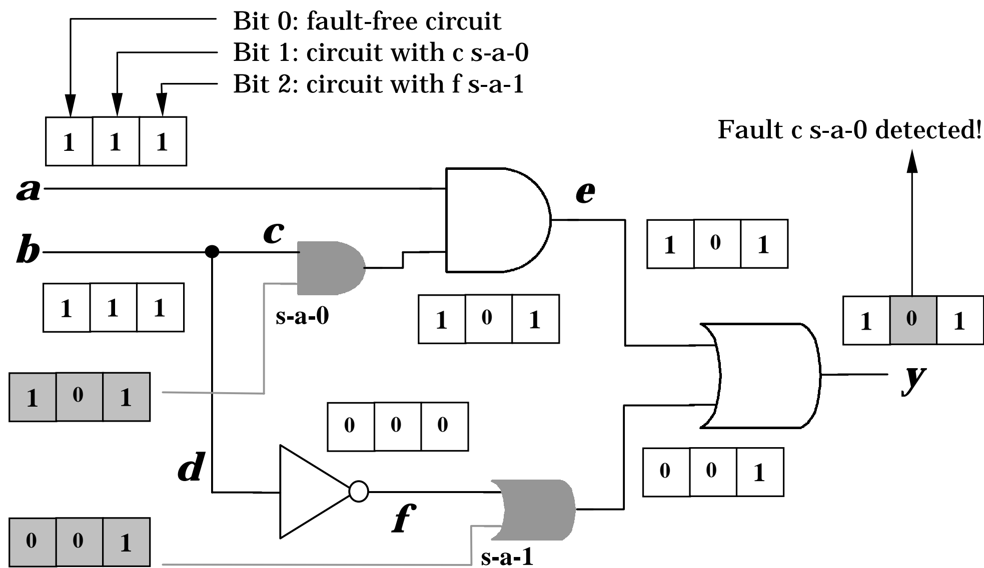


Figure 4.6. Fault injection in parallel fault simulation

In parallel fault simulation stuck at faults are injected to the circuit as follows. A stuck-at one fault at a line is injected by ORing a word, called a *fault mask*, containing a 1 in the bit position corresponding to the faulty value and 0s elsewhere to the word representing the signal values at the line. Similarly, a stuck-at zero fault can be injected by ANDing a word

with a 0 in the faulty value position and 1s elsewhere. Figure 4.6 shows how the faults can be injected in the parallel fault simulation by adding dedicated gates and constant inputs that would serve as bit masks.

The basic idea of parallel fault simulation on SSBDD models is similar to this type of simulation at gate-level. Here, traditional style SSBDD descriptions are preferable since in these we do not have to keep track of whether a variable labeling a node is inverted or not (See Subsection 2.5 for explanation on SSBDD description styles). Figure 4.7 presents parallel fault simulation on an SSBDD corresponding to the circuit in Figure 4.6, where the faults have already been injected. The simulation takes place as follows. Starting from the node with the highest index value, we repeat operation:

$$\left(x(m)\ \&\ x(m^1)\right) \vee \left(\overline{x(m)}\ \&\ x(m^0)\right)$$

for each node of the SSBDD. In this operation, m denotes the current node and m^0 and m^1 are its 0- and 1-successors, respectively. x(m) denotes value of the variable labeling the node m. The result of the simulation will be the value calculated for the root node m_0.

However, two logic values are usually not sufficient for accurate logic simulation. At the very least, a third value, usually denoted by x or u, is needed in order to represent unknown signal values. In parallel fault simulation, coding techniques are required when more than two logic values are used.

Two bits are necessary for representing each signal value in three-valued logic simulation. Two words will be used for representing a set of fault-free and faulty values. Let us denote these words by W1 and W2. A pair of bits, one from the same position in each of the two words will represent each logic value. This coding is similar to that used in [Chap74]. Table 4.4 shows coding that can be used for representing three values.

Table 4.4. Coding for three-valued logic in parallel simulation

W1	W2	value
0	1	0
1	0	1
1	1	X
0	0	unused

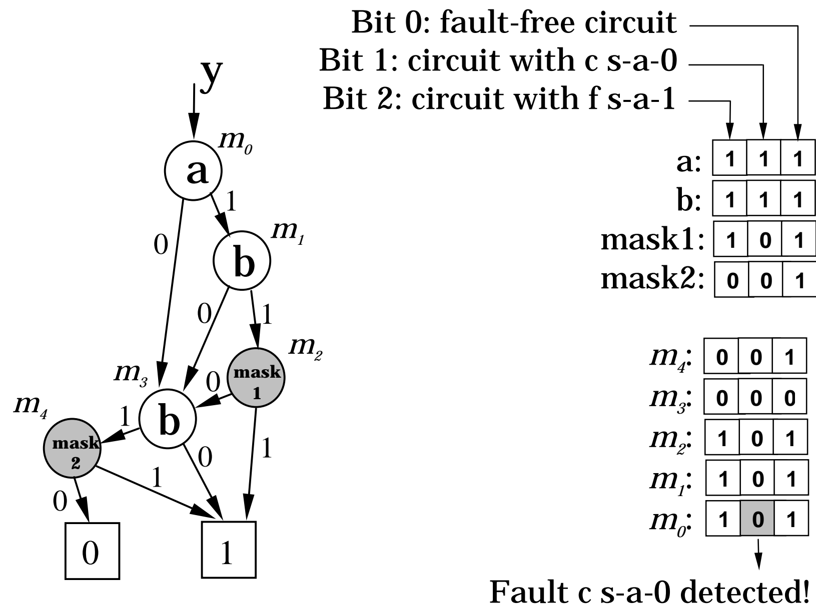
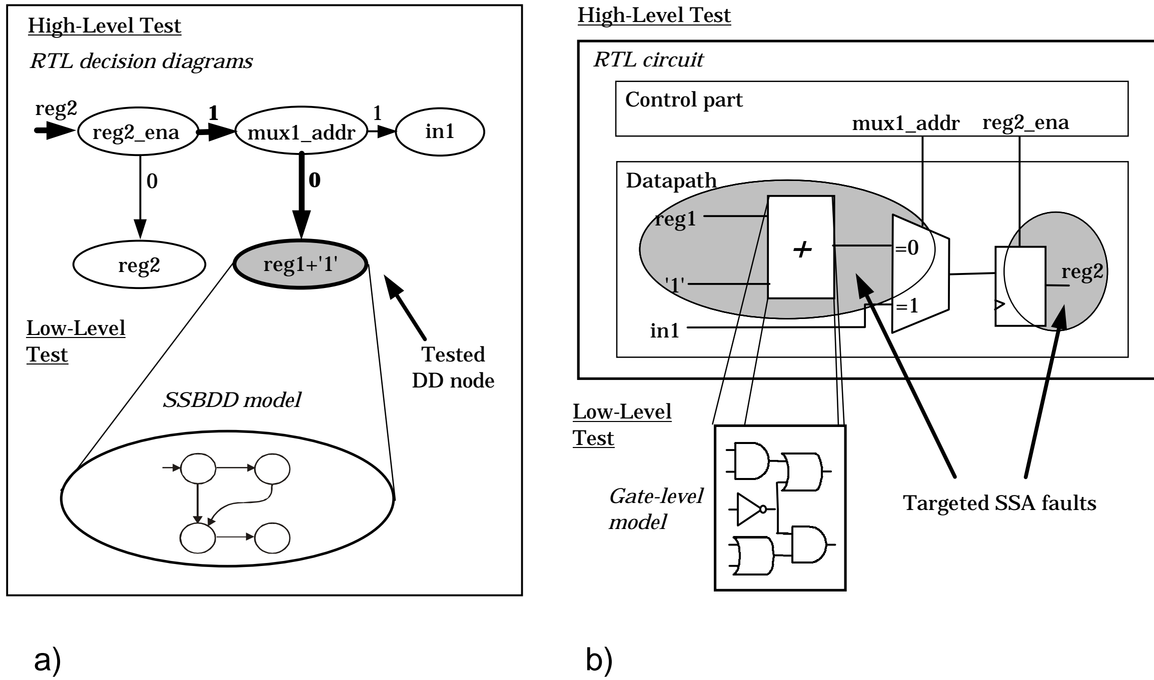


Figure 4.7. Parallel fault simulation on SSBDD representations

Table 4.5 presents the operations performed with words W1 and W2 which code the gate input values. Here, A_1 represents the word W1 of the line corresponding to the first input of the logic gate, and B_1 denotes the W1 value for the second gate input. Similarly, A_2 denotes the W2 value for the first gate input and B_2 is the corresponding value for the second gate input. For AND gate the W1 words of gate inputs have to be ANDed and W2 words ORed, while for OR gate the W1 words of inputs have to be ORed and W2 words ANDed. Inversion operation is performed by simply switching the words W1 and W2.

Table 4.5. Operations for logic gates in three-valued parallel simulation

gate	W1	W2
AND	A_1 & B_1	$A_2 \vee B_2$
OR	$A_1 \vee B_1$	A_2 & B_2
INVERTER	A_2	A_1



a) b)

Figure 4.8. Example of scanning test.

Table 4.6 shows the operations performed with nodes W1 and W2 in SSBDD representations. Similarly to two-valued parallel simulation on SSBDDs, we start from the node with the highest index value and move torwards the root node. In this operation, m denotes the current node and m^0 and m^1 are its 0- and 1-successors. x(m) denotes value of the variable labeling node m. The result of simulation will be the value calculated for the root node m_0. Thus, it can be concluded that SSBDD representations are feasible models for both, 2- and 3-valued parallel fault simulation.

Table 4.6. Operations for SSBDD nodes in three-valued parallel simulation

W1	W2
$\left(x(m) \,\&\, x(m^1)\right) \vee \left(\overline{x(m)} \,\&\, x(m^0)\right)$	$\left(x(m) \,\&\, x(m^0)\right) \vee \left(\overline{x(m)} \,\&\, x(m^1)\right)$

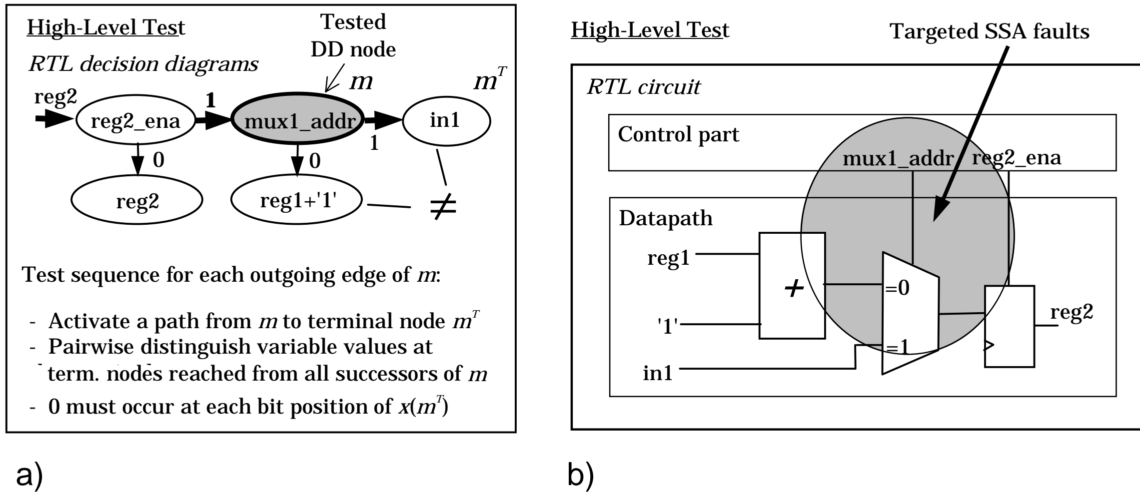


Figure 4.9. Example of conformity test.

4.4 Hierarchical Fault Model for Multi-Level Decision Diagrams

In the hierarchical fault model presented in this thesis appropriate tests for the corresponding nodes of the RT-level decision diagram model have to be set up. There are two types of tests, refered to as *scanning test* and *conformity test*, respectively. Scanning tests are applied to terminal nodes of RTL decision diagrams and their aim is to test the functional units (FU), registers and constant busses of the datapath. In the case of testing FUs, the low-level description (SSBDD model) of the module is needed in order to generate test at this level. For this purpose, fault modeling techniques presented in above subsections are implemented.

Figure 4.8a presents an example of a DD, where a scanning test is set up for a terminal node labeled by addition operation. In order to perform scanning test for this node, path to it is activated in the DD. Thus, the variable *reg2_ena* is set to 1 and *mux1_addr* is set to 0. Subsequently, the adder module corresponding to the terminal node under test is tested at the low-level (i.e. SSBDD model). Figure 4.8b shows the RTL subnetwork represented by the DD in Figure 4.8a. The grey circles indicate areas of the network covered by this scanning test.

Conformity tests are set up for non-terminal nodes and they target the decoding logic of the multiplexers of datapath as well as the output logic of the control part. Conformity test is similar to scanning test in the way that a path is activated to the node under test. In addition, distinguishing of values of the variables labeling the terminal nodes is made. In current implementation pairwise distinguishing is used. Conformity test for a node must be carried out for each edge of the node under test to be activated and for each pair of the successors to be distinguished. Hence, there exist n(n-1) conformity tests for a non-terminal node with n successor nodes. In case the successor nodes are not terminals nodes, two terminal nodes are chosen and paths are activated from the successors to these nodes.

There exist some additional, implementation-specific criteria included to conformity tests. For example, in the case of AND-OR type multiplexers that most of the VLSI technologies are implementing, the set of conformity tests for a node should confirm to the following criterion. It was mentioned above that conformity tests are carried out for each outgoing edge of the node under test to be activated. In the distinguishing test for each such activated edge, values of the variable labeling the terminal node at the end of the main activated path must cover zeros in all bit positions. Several tests may be required in order to reach this goal for a single pair of successor nodes to be distinguished.

A simple example in Figure 4.9a illustrates the conformity test for the control signal *mux1_address*. Here, a path is activated to the node labeled by *mux1_address*, setting *reg2_enable* to 1. Subsequently, two conformity tests for this node are performed. The first one with *mux1_address* being 0 and the second one with *mux1_address* equal to 1. In both tests the values of *in1* and *reg1* + 1 must be different (i.e. distinguished). In addition, in the first test, 0s must occur in each bit position of the result of *reg1* + 1. (Note, that in general case, several tests are needed to fulfill this requirement). Similarly, in the second test, 0s must occur in each bit position of *in1*. Figure 4.9b shows the subnetwork represented by this DD. The corresponding areas of the subnetwork, where SSA faults are targeted by the test are marked with grey areas.

Note, that in general case, there is no one-to-one correspondence between separate DD nodes and modules in the RTL network. However, if we test all the nodes of a DD, we will consequently test all the SSA faults in the

corresponding subnetwork of the circuit. This is a very important feature of the presented hierarchical fault model.

4.5 Conclusions

In this Chapter a novel hierarchical fault model was presented. At the low-level the circuit is modeled by Structurally Synthesized BDD (SSBDD) representations. We have pointed out that SSBDDs have a number of beneficial properties that make them better suitable for modeling single stuck-at faults than traditional logic level representations. Furthermore, it was shown that SSBDDs are a feasible circuit model for both, two- and three-valued parallel fault simulation. Finally, a hierarchical fault model was proposed, where using a combination of dedicated tests for the terminal and non-terminal nodes, 100 % stuck-at coverage for the circuit datapath is achieved. Although a part of the control part is targeted by these tests, as a future work, this model could be extended to include all the nodes of control part as well.

5 Hierarchical Test Generation on Multi-Level Decision Diagrams

While automatic test pattern generation for combinational circuits was considered to be a solved problem already by the end of 1980s, test generation for sequential circuits still remains a major challenge. There have been many different approaches to sequential circuit test proposed over the years. Despite of the diversity of techniques offered to solve the problem, the achieved fault coverages tend to be unsatisfactory and test generation times long for more complex circuits. As a possible solution, hierarchical approaches implementing information from several (usually two) abstraction levels have been proposed. Current Chapter presents a novel hierarchical method for sequential circuit testing based on multi-level decision diagram models. Experiments show that the proposed method allows to reach high fault coverages for circuits with complex sequential structures in a very short time.

5.1 Introduction

As the degree of integration in VLSI designs has been growing, so has the need for automation of different design tasks. Design automation helps to shorten the time-to-market cycle and improves significantly designer's productivity. The automation was first introduced on the lower levels of design tasks, like placement and routing, and together with the growth of design complexities, moved gradually to higher levels, e.g. logic synthesis, high-level synthesis (HLS) and hardware/software co-design. Nowadays the goal is to automate the entire design cycle from conceptualization to generation of silicon layout.

During recent years, more-and-more commercial and university high-level synthesis tools [Gajs89] have become available. These tools are applied for automatically generating a register-transfer level (RTL) description from a behavioral description of the circuit. In the RTL descriptions the design has been partitioned into a control part, i.e. a finite state machine, and a datapath part containing a network of interconnected functional units (FU). Usually the HLS tools take into account several constraints, as

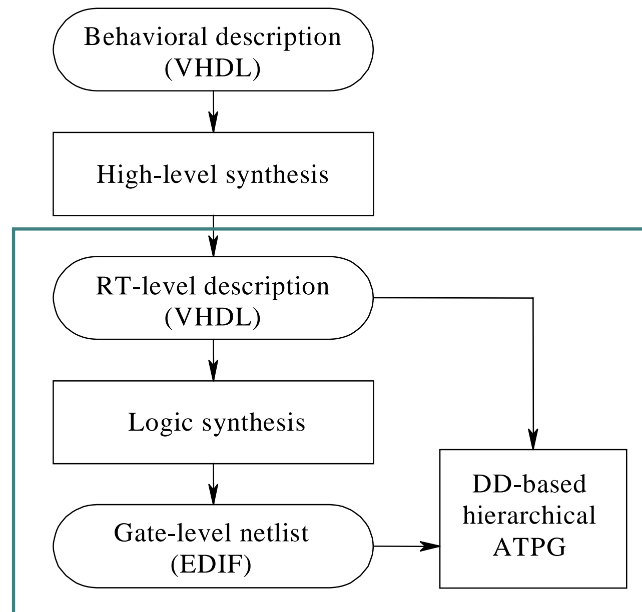


Figure 5.1. Design and test flow

speed, area, or testability [Flot95, Bhat94], and allow the designer to quickly compare the trade-offs between alternative RTL implementations.

The circuit design flow continues by synthesizing the RTL descriptions to the logic level. Subsequently, the logic gates are placed to the chip layout and connections between them are routed. Finally, the chip is manufactured in a silicon foundry and tested.

Let us now see how the hierarchical test generation system presented in this thesis does fit into the circuit synthesis flow described above. Figure 5.1 shows the place of the system in the general design flow. The environment consists of a hierarchical test generator and dedicated high- and low-level DD model interfaces. From RT-level VHDL descriptions, high-level DD interface generates RTL DD models, which are applied as the high-level input for the test generator. Structurally Synthesized BDD representations are required when generating local, structural level tests for the functional units (FU) of the design. For that purpose, an EDIF to SSBDD interface has been implemented. The system includes also a low-level sequential circuit fault simulator, which is needed to measure the stuck-at fault coverage of the tests generated by the hierarchical ATPG.

In hierarchical testing, top-down and bottom-up strategies are known. In the bottom-up approach [Mur88], tests generated at the lower level will be later assembled at the higher abstraction level. Such algorithms ignore the incompleteness problem: constraints imposed by other modules and/or the network structure may prevent test vectors from being assembled.

Top-down approach [Lee94] was introduced to avoid this problem by deriving global constraints for low-level solutions. Current thesis presents a method which is based on the latter approach, where constraints are extracted at the higher level with the goal to be considered when deriving tests for modules at the lower level.

Previous works in the area of hierarchical testing have the following main shortcomings:

3. Only the faults in the datapath Functional Units (FU) are targeted.
   This usually results in low fault coverages for the control part as well as for multiplexers, registers and fanout buses of the datapath.

4. Complex symbolic algebra is used in high-level path activation.
   This adds computational overhead to test generation and, in some implementations, it can also cause loss of solutions due to conflicting symbol assignments while activating the high-level symbolic paths.

The aim of the approach proposed in current thesis is to overcome the above mentioned shortcomings. The speed-up of the test generation technique is achieved by implementing simplified fault propagation along single path, and constraint justification, where transparency rules are neglected. The simplification of fault propagation may, in some cases, lead to a decrease in fault coverage because potential fault masking is not considered. However, our experiments have not so far indicated any decrease in fault coverage caused by fault-masking in propagation.

The approach is based on using Decision Diagram (DD) models where, differently from known methods, control unit and datapath are handled in a uniform manner. RT level model of the control and datapath parts is represented by high-level DDs, whereas the gate-level descriptions of RTL blocks are given by Structurally Synthesized Binary Decision Diagrams. The method proposed in the paper combines deterministic and simulation-based techniques for test pattern generation. On the RT-level,

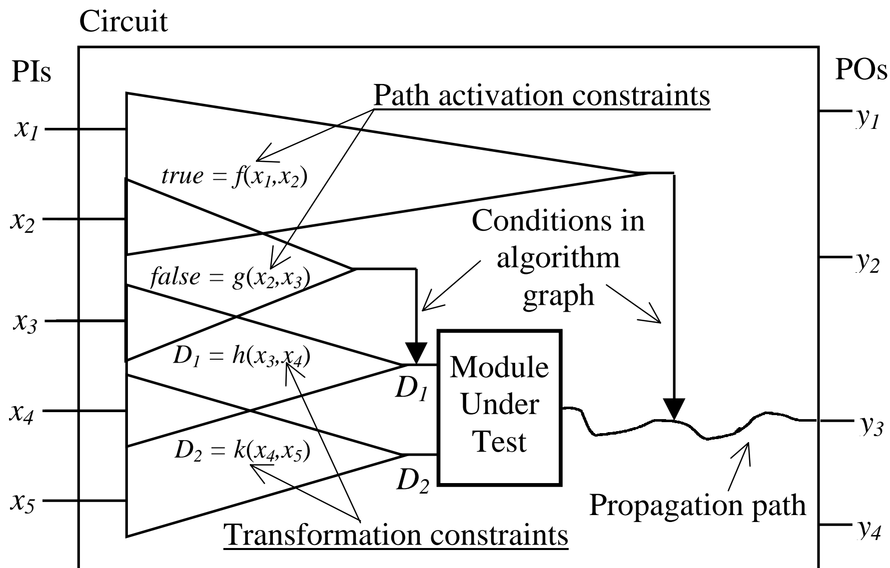


Figure 5.2. High-level test generation constraints

deterministic path activating is mixed with simulation-based techniques used in constraints solving. The gate-level local test patterns for components are randomly generated driven by high-level constraints and partial path activation solutions.

We will continue our talk about the DD-based hierarchical test generation algorithm as follows. Subsection 5.2 introduces the concept of high-level test generation constraints. In Subsection 5.3 the hierarchical test generation algorithm is presented. Subsection 5.4 explains the algorithm on a simple example. Finally, experimental results and conclusions are given.

5.2 Concept of High-Level Test Generation Constraints

The high-level test generation constraints considered in current approach are divided into two categories: *path activation constraints* and *transformation constraints*. Path activation constraints correspond to the logic conditions in the algorithm graph flow that have to be satisfied in

order to perform propagation and value justification through the circuit. Extraction of this type of constraints from conditions traversed in the circuit algorithm is explained basing on an example in the following subsection.

Transformation constraints, in turn, reflect the value changes along the paths from the inputs of the high-level module under test to the primary inputs of the whole circuit. These constraints are necessary in order to calculate the local test patterns for the module under test.

Figure 5.2 explains the role of these two types of constraints in test generation for a circuit module. In the Figure there are two path activation constraints: true = $f(x_1,x_2)$ and false = $g(x_2,x_3)$. The first one is necessary to propagate the value from the output of the module to the primary output y_3 of the circuit. The latter is required for justification of the first input (D_1) of the module under test. Both these constraints are extracted from the conditional nodes traversed in the algorithm graph flow of the circuit. (See section 5.3.3 for the concept of algorithm graph flow).

In addition, the Figure presents two transformation constraints. These constraints represent the function for computing what will be the value of the corresponding module input depending on the values of primary inputs of the circuit.

Both types of constraints can be represented by common data structures and manipulated by common procedures for creation, update, modeling and simulation. In the following, the data structure and update operations of high-level test generation constraints are defined.

<u>Definition 5.1</u>: A condition C = g(x), where C is an integer, Boolean or symbolic value, and g(x) is an expression on a subset of variables of the DD model representing the circuit under test, is refered to as constraint.

In current approach, symbolic values that can be used for C in a constraint C = g(x) are D_i, which correspond to the local test patterns for the i-th input of the current Module Under Test (MUT) in the datapath (See Figure 5.2).

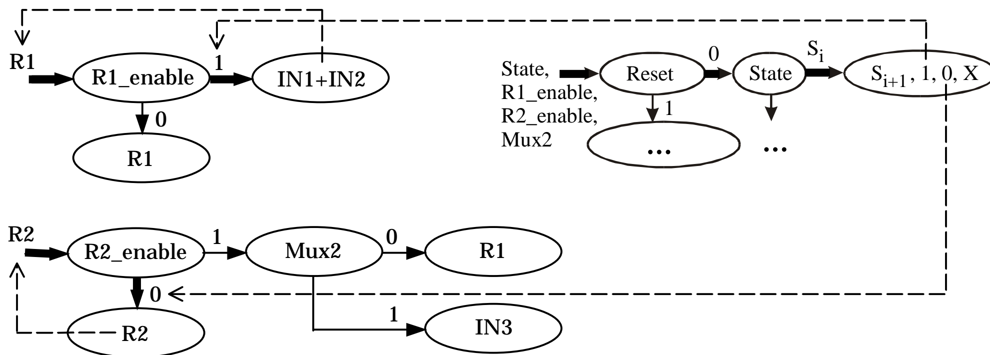


Figure 5.3. Substituting datapath variables during constraint update

Definition 5.2: Constraint C = g(x) is said to be *justified* if $x \subseteq x_I$ and is called *unjustified* otherwise. Here, x_I denotes the set of input variables of the DD. (See Subsection 2.3 for explanation on variable types in the RTL DD representations).

If a constraint C = g(x) is unjustified then all the variables $x' \subseteq x$ that are not input variables x_I are said to be *unjustified variables* of the constraint.

Definition 5.3: Let x' be the set of justified variables and x" the set of unjustified variables of a constraint C = g(x',x"). Let Gx''_i be the graph calculating the value for $x''_i \in x''$. Let us assume that in every Gx''_i there exists an activated full path to a terminal node m^T_i. The process, where each variable x''_i is substituted by $x_i'''=x(m^T_i)$ is refered to as *updating the constraint*. Updating the constraint C = g(x',x") by replacing variables x''_i with corresponding subexpressions $x(m^T_i)$ creates a new constraint C = g'(x',x'''), where g' can be regarded as a superposition of functions on a set of variables in the DD representation.

Figure 5.3 shows how the variables in the constraints are substituted during the constraints updating process. It depicts a control part DD and two datapath DDs, which calculate the values for datapath variables R1 and R2. The terminal node selected from the control part DD determines value assignments to control signals: R1_enable=1 and R2_enable=0. These assignments in turn activate full paths in datapath DDs. Variables labeling the terminal nodes at the end of these paths will substitute variables corresponding to respective DDs in all the test generation

constraints. In our example, variable R1 is substituted by expression IN1+IN2 while variable R2 is not substituted.

Note that there are no decisions included to the variable substitution during constraint update. Constraints extracted by update as defined above are free of control variables x_C since terminal nodes of DDs are not labeled by x_C. They are also free of cycles as all cycles are unfolded by superposition. The process of constraint updating is further explained in Figure 5.14 in Subsection 5.3.4.

5.3 Hierarchical Test Generation Algorithm

5.3.1 Introduction

This subsection explains the hierarchical test generation algorithm based on multi-level decision diagram models. First, we explain the basic concepts of the algorithm, then follows the top-level view of the different stages. Finally, each algorithm stage is explained in detail.

In the hierarchical algorithm, the values that can be assigned to high-level DD model variables are 'X' (don't care), integer or Boolean. In current implementation, we have neglected the technique of propagating the fault effect as a symbol. The reason is in avoiding the time consuming procedure of determining the fault effect frontier during each step of the symbolic path activation. Instead we use a pointer to the variable, where the fault effect has been propagated and denote it by p_D.

Test generation for a component Module Under Test (MUT) represented by a node m in graph G_y begins with the *fault manifestation* procedure. Two methods are used: scanning test generation (for terminal nodes) and conformity test generation (for nonterminal nodes). During the fault manifestation phase in graph G_y, fault effect pointer is set to point to variable y. Let us denote it by $p_D \rightarrow y$. In addition, symbolic transformation constraints are created in form of $D_i = x_i$, i = 1,…,n, where n is the number of variables in the expression x(m) i.e. the number of inputs of current MUT. D_i are symbolic values representing local test patterns applied to corresponding inputs of MUT and x_i are the variables of DD model representing these inputs.

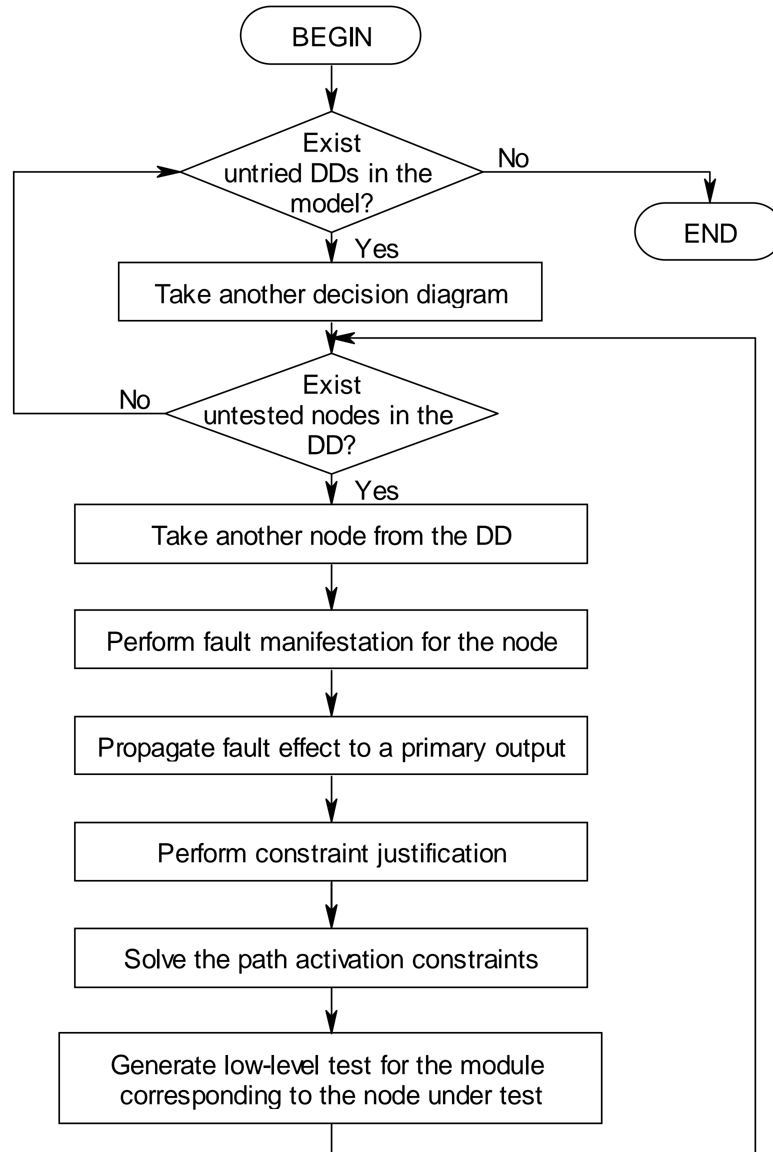


Figure 5.4. The general flow of the hierarchical test generation algorithm

Fault manifestation is followed by *fault effect propagation*. During the propagation stage we move forward in time (clock-cycles), fault effect is propagated towards primary outputs and path activation constraints are created whenever conditions in the control part DD are traversed. Propagation is completed when the fault effect pointer points to variable x corresponding to a primary output of the circuit, i.e. $p_D \rightarrow x : x \in x_O$.

Subsequent to propagation, *constraint justification* starts. Justification moves backwards in time, starting from the clock-cycle, where propagation ended. During this process existing constraints are updated and additional path activation constraints are created. Finally, *constraints solving* procedure is applied to the extracted constraints and MUT is fault simulated by constraint-driven, randomly generated local test data. The general test generation flow is presented in Figure 5.4, and the different stages are explained in detail in following subsections.

5.3.2 Fault manifestation

As it was mentioned above, appropriate tests for the corresponding nodes of the RT-level DD model have to be set up during the manifestation stage. The two types of tests are referred to as *scanning test* and *conformity test*, respectively. Scanning tests are applied to terminal nodes and their aim is to test the functional units (FU), registers and constants of the datapath. Conformity tests are set up for non-terminal nodes and they target the decoding logic of the multiplexers of datapath as well as the output logic of the control part. The basic concept of setting up these tests in a DD were presented in Chapter 4. In the following we will explain how to apply these tests in fault manifestation stage of the hierarchical test generation algorithm.

During the scanning test, the path to the node under test is activated in respective DD. Fault-effect pointer y_D is pointed to the variable corresponding to the DD, and transformation constraints $D_i=x_i$ are created, where x_i is the i-th argument of the function corresponding to the module under test.

Conformity test is similar to scanning test in the way that the path is activated to the node under test, and the fault effect pointer is set to the DD variable. During the manifestation stage of the conformity test transformation constraints $D_i=x(m^T_i)$ are created, where m^T_i is the terminal node at the end of the path that is activated from the i-th successor of the node under test. Since current approach uses pairwise distinguishing of successor nodes, exactly two such constraints are created.

Note, that the symbolic values D_i are treated differently in the final stage of the test generation for a DD node. In the case of scanning test these

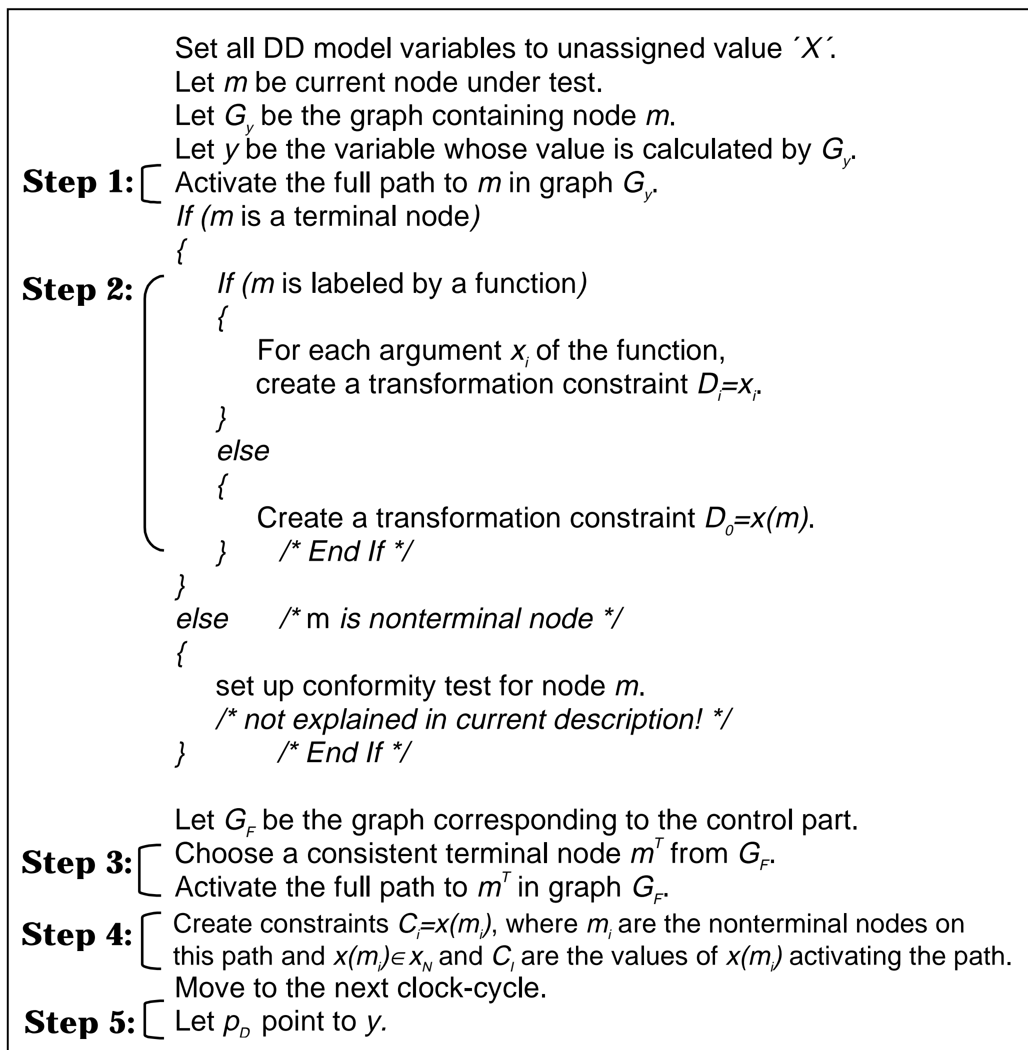


Figure 5.5. Algorithm for fault manifestation

symbols represent local test vectors to be applied to the inputs of the FU under test for fault simulation purposes. For conformity test the symbols represent the variable values at the terminal nodes of the DD that have to be distinguished according to the criteria defined in Chapter 4. All the remaining stages of the algorithm (i.e. fault effect propagation, constraint justification and solving of path activation constraints) for the two types of tests are exactly identical.

Additional difference between the two types of tests is that in setting up

the conformity test, decisions are made. Generally there exist multiple terminal nodes to which a path can be activated from a successor node of the node under test. Due to this fact there is a choice and one of the terminal nodes is selected while the others are marked as untried selections in the decision-tree. They will be selected later if with the first terminal node we are not able to generate the test and backtracks in the algorithm lead us to this decision-tree node.

Prior to the algorithm description, we introduce the definition of *consistent terminal node*.

Definition 5.4: Let m^T be a terminal node of a DD G_y calculating the value for variable y. If current value assignments are consistent with the full activated path to m^T and the value of y is consistent with the value of $x(m^T)$ then it is said that m^T is a *consistent terminal node* of G_y.

In Figure 5.5, the algorithm description of fault manifestation stage is given. For the sake of simplicity, we have presented the algorithm as for the register-oriented RODD model (see Chapter 2.3) and omitted the conformity test specific manifestation details from the description.

Let us explain the above-described algorithm basing on a simple example in Figure 5.6, where our aim is to set up scanning test for the node labeled by operation J+K in a datapath DD. During that process, the following steps (also shown in the description in Fig. 5.5) have to be performed.

1. Activate the full path to the node under test.
   In current example, scanning test for the node labeled by operation J+K is set up. The path from the root node to the corresponding terminal node is activated, triggering variable assignments A=1, B=1.

2. Create transformation constraints.
   Transformation constraints D_1=J and D_2=K are created.

3. Select the FSM state for manifestation.
   While selecting the FSM state for manifestation, a terminal node of the control part DD is chosen, whose variable vector values are consistent with the values of corresponding control signal x_C assignments performed during step 1 of the setup. There exists always at least one such node. In this example, the chosen next FSM state is

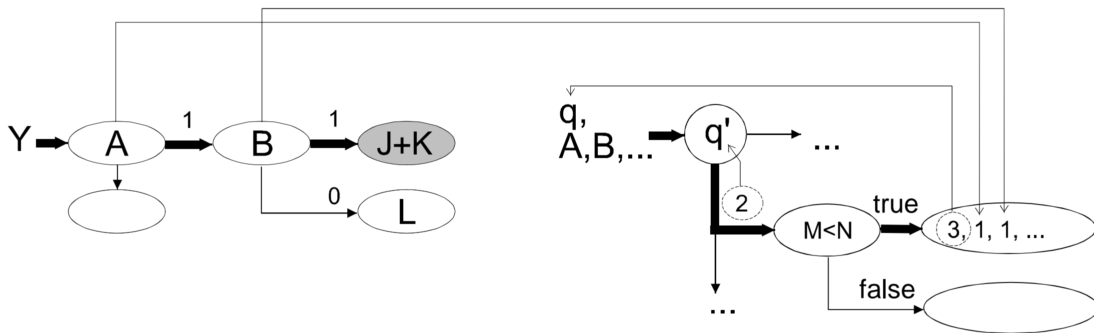


Figure 5.6. Fault manifestation for node labeled by J+K

state 3. The current manifestation state is equal to the label of the activated edge of the node labeled by the state variable q. Thus, current manifestation state is 2.

4. Create path activation constraints.
   As the path is activated in the control part DD, constraints are created for all the non-terminal nodes labeled by conditional signals x_N originating from the datapath. In our example, a path activation constraint true=M<N is created.

5. Fault effect pointer is set to the variable calculated by current DD.
   The pointer is set to point to the variable whose value is calculated by current DD G_Y. In other words $p_D \rightarrow Y$.

When a test for a DD node is set up, the fault effect has to be propagated to a primary output of the circuit and the created constraints have to be justified and solved. These tasks will be explained in the following.

5.3.3 Fault effect propagation

The fault manifestation stage presented above sets the fault effect to the output of the RTL Module Under Test (MUT) and determines current FSM state. The goal of fault propagation procedure is to calculate a state sequence required to propagate this fault effect from MUT to a primary output of the device. During this process, path activation constraints are created of the conditions traversed in the control part DD. In Figure 5.7 the algorithm for fault effect propagation is presented.

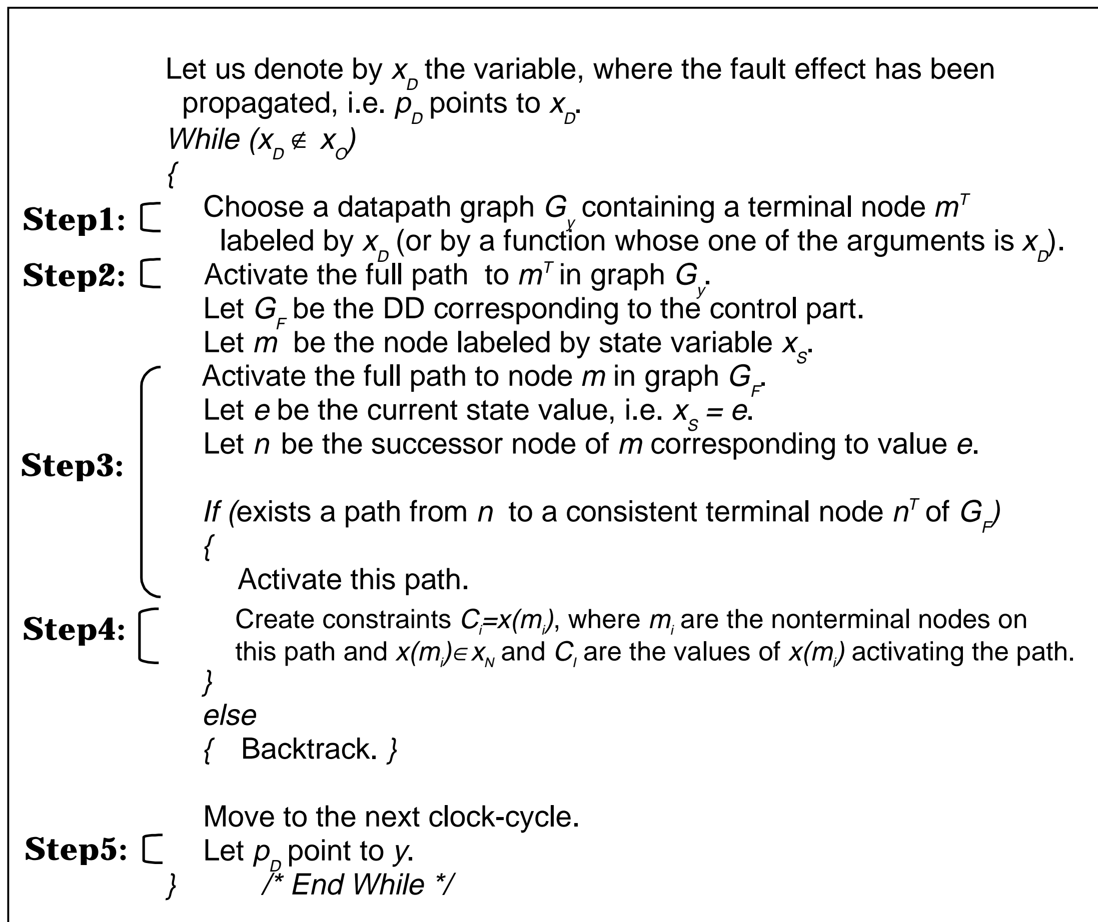


Figure 5.7. Algorithm for fault effect propagation

The fault propagation algorithm described in Figure 5.7 has a simple structure. No transparency rules for modules are applied and no exact calculation of the fault effect value is performed during the process. This allows us to avoid the detailed justification of values, which otherwise have to be performed in the datapath in order to create transparent paths through components. The task of justifying such values may lead to a high number of backtracks because of frequent conflicts inherent in datapaths of complex sequential structure.

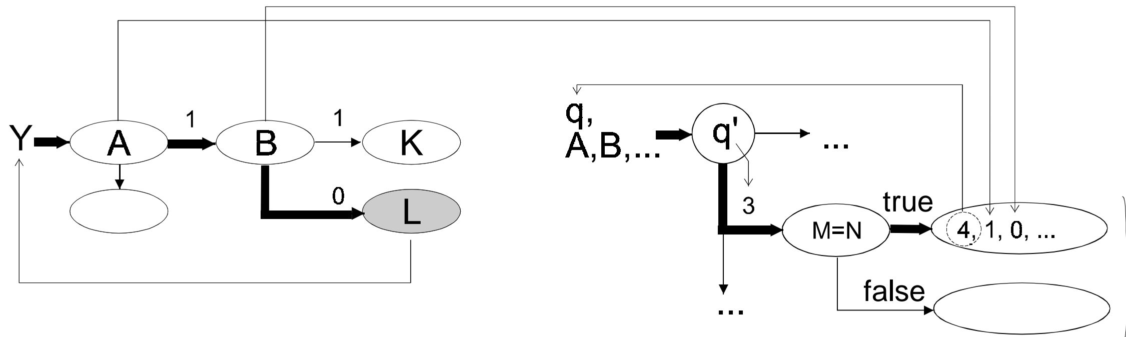


Figure 5.8. Fault effect propagation on RTL DD models

Figure 5.8 explains propagation on RTL decision diagram representations. The objective of the example is to propagate the fault effect from variable L towards primary outputs, when current state is 3. In order to achieve that goal, the following steps are required. (The corresponding steps are denoted additionally in the algorithm description in Figure 5.7).

1. Select a graph with a node m, where $p_D \rightarrow x(m)$.

   Not shown in Figure 5.8, where this graph has already been selected. See Figure 5.7.

2. Activate the full path to the node labeled by fault effect.

   In current example, the fault effect pointer points to variable L. The path from the root node to the corresponding terminal node is activated, triggering variable assignments A=1, B=0.

3. Select the next FSM state.

   While determining next state, a terminal node of the control part DD is chosen whose variable vector values are consistent with the values of corresponding control signals x_C assignments performed during step 2. Differently from determining next state during manifestation, the candidates for the terminal nodes are constrained by the current state value. In the example, current state value is 3 and the chosen next state is 4.

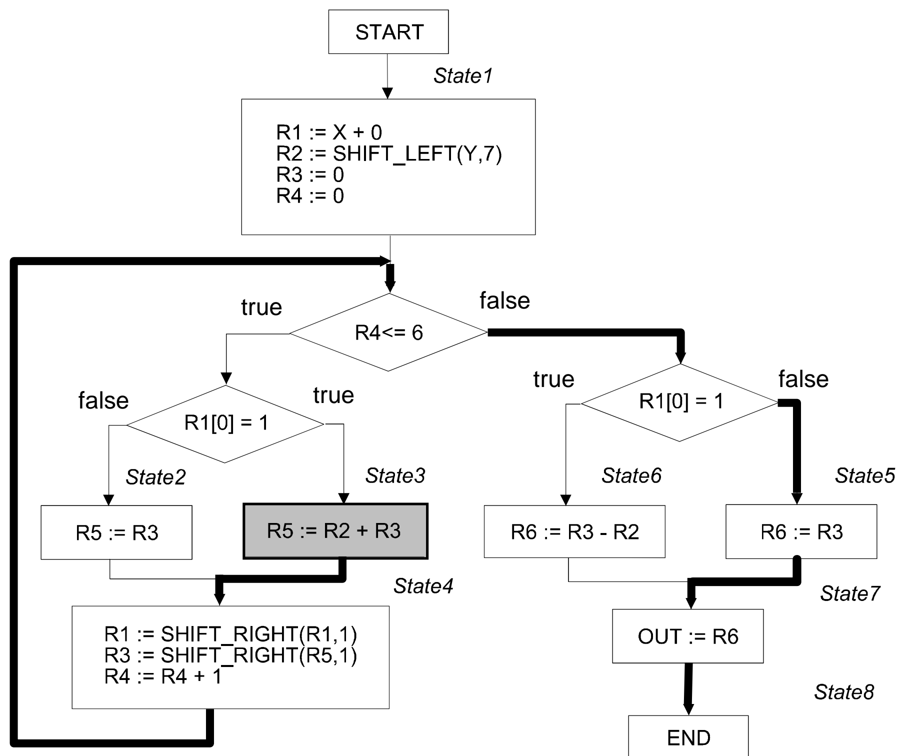


Figure 5.9. Propagation on algorithm graph flow of mult8x8

4. <u>Create path activation constraints.</u>

   As the path is activated in the control part DD, constraints are created for all the non-terminal nodes labeled by conditional signals x_N originating from the datapath. In our example, a path activation constraint true=(M=N) is created.

5. <u>Fault effect pointer is set to the variable calculated by current DD.</u>

   The pointer is set to point to the variable whose value is calculated by current DD G_Y. In other words $p_D \rightarrow Y$.

Previously we discussed how the propagation procedure is performed technically on the DD representations. Let us now consider a more high-level view of this process. Behavior of an RTL design can be described by means of Algorithm Flow Graph (AFG). AFG is a directed graph, which can be represented by a pair G=(N,A), where N is a set of nodes and A is a set of directed arcs between the nodes. $N=C \cup O \cup S \cup E$, where C is a set

of conditional nodes, O is a set of operator nodes, S is the starting node and E is the ending node. The algorithm starts from node S and terminates in E. According to the result of the operation in conditional node c, the arc starting from c labeled by corresponding value will be selected. Operator nodes are labelled by operations, which can be executed simultaneously in a single clock cycle. From this type of nodes, always only a single arc starts. In the case of Moore automata, operator nodes are additionally labeled by states, in Mealy automata the labels are added subsequent to operator nodes, respectively. Figure 5.9 shows the AFG corresponding to the RTL description of a multiplier circuit mult8x8. In the example, Moore automaton is implemented. The circuit has only one observable output OUT and two primary inputs X and Y.

Figure 5.9 explains the propagation procedure for the addition operation R2+R3. The propagation state sequence is denoted by bold arrows in the Figure. As it can be seen, the fault effect is set up in register R5 (at state3), propagated via R3 (state4) and R6 (state5) to a primary output OUT (state7). During the propagation sequence two conditional nodes are traversed creating two path activation constraints: false=R4≤6 and false=(R1[0]=1).

5.3.4 Constraints justification

As it was mentioned above, justification process starts from the clock-cycle, where propagation finished. Justification traverses backwards the state sequence calculated by propagation phase until the clock-cycle of fault manifestation is reached. During this process, constraints previously created by propagation are updated. Starting from the clock-cycle of manifestation phase, a reverse state sequence is calculated, existing constraints are updated and additional constraints are created of the conditions in FSM that have to be satisfied. Note that at each clock-cycle, from all the created constraints only those are considered during justification that were created in a later clock-cycle than current one.

During the clock-cycles earlier than the manifestation, each justification step consists of three consecutive stages. These are selection of current objective, justification of the objective and updating of existing constraints, respectively. Constraint updating is explained at the end of current section, basing on an example.

```
        While (exist constraints with unjustified variables)
        {
          If (current clock-cycle is earlier or equal to that of manifestation)
          {
Step1:  [   Let current objective be to justify variable y.
            Let G_y be the graph that calculates the value of y.
Step2:  [   Choose a consistent terminal node m^T from G_y.
        [   Activate the full path to m^T in graph G_y.

            Let G_F be the graph corresponding to the control part.
            If (exists consistent terminal node n^T in G_F)
            {
Step3:  [     Activate a full path to n^T in graph G_F.

Step4:  [     Create constraints C_i=x(m_i), where m_i are the nonterminal nodes on
        [     this path and x(m_i)∈x_N and C_i are the values of x(m_i) activating the path.

            }
            else
            { Backtrack. }
          }   /* End If */

Step5: [  Update the constraints created later than current clock-cycle.
          Move to the preceding clock-cycle.
        }       /* End While */
```

Figure 5.10. Algorithm for constraint justification

Figure 5.10 presents the algorithm for symbolic constraint justification. Figure 5.11 explains justification on RTL decision diagram representations. The objective of the example is to select current state and to perform justification for variable Y, when the next state is 2. In order to achieve that goal, the following steps are required. (The corresponding steps are denoted additionally in the algorithm description in Fig. 5.10).

1. <u>Determining current justification objective.</u>
   This step is not explained in Figures 5.10 and 5.11. Current justification objective is found as follows. If there exist unjustified variables in transformation constraints then justification objective will be to justify first such variable. Otherwise, current justification objective will be to justify the first unjustified variable in path

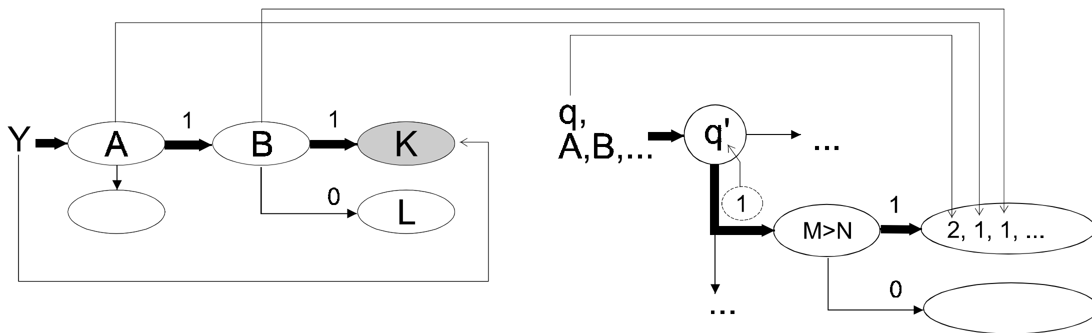


Figure 5.11. Justification for variable Y

activation constraints. When there are no constraints left containing unjustified variables, justification ends.

2. Activate the full path to a terminal node in the DD of justification objective.

In current example, justification objective is to justify variable Y. We choose a terminal node (labeled by K) and activate the path to it, triggering variable assignments A=1, B=1.

3. Select present FSM state.

While determining present state, a terminal node of the control part DD is chosen, whose variable vector values are consistent with the values of the next state and corresponding control signals x_C assignments performed during step 2. In the example, the chosen present state is 1.

4. Create path activation constraints.

As the path is activated in the control part DD, constraints are created for all the non-terminal nodes labeled by conditional signals x_N originating from the datapath. In our example, a path activation constraint true=(M>N) is created.

5. Update the constraints.

All the constraints, which were created in a clock-cycle later than the current clock-cycle are updated. This is done according to the new values assigned to the control variables x_C determined by the selected terminal node of the control part DD. These values create activated

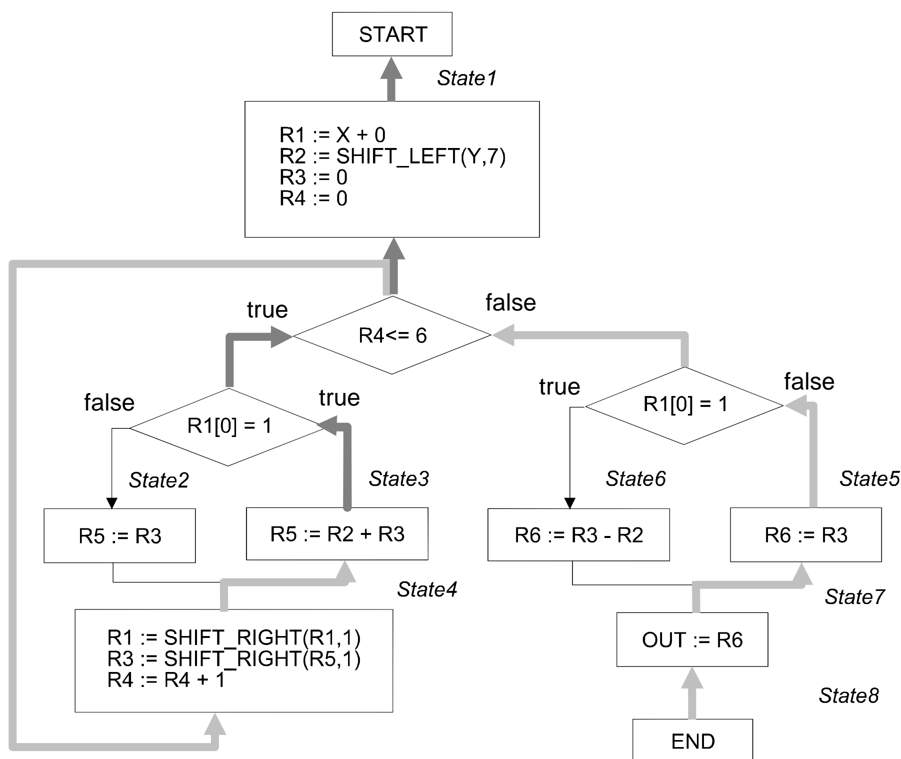


Figure 5.12. Justification on an algorithm graph flow.

paths in datapath DDs, which lead to some terminal nodes. The corresponding variables of all the graphs will be substituted by the values labeling the corresponding terminal nodes in all the constraints. (See Subsection 5.2 for explanation of the updating process).

As a partial result of updating process, variable Y in all the constraints is substituted by K. In our example the entire process has not been explained since we are considering only a single datapath DD, not a system of DDs.

Figure 5.12 explains the justification procedure for the mult8x8 example algorithm graph flow presented in Figure 5.9. The state sequence traversed during justification is denoted by bold arrows in the Figure. As it can be seen, during the justification sequence two conditional nodes are traversed in addition to the propagation creating two new path activation constraints: true=R4≤6 and true=(R1[0]=1).

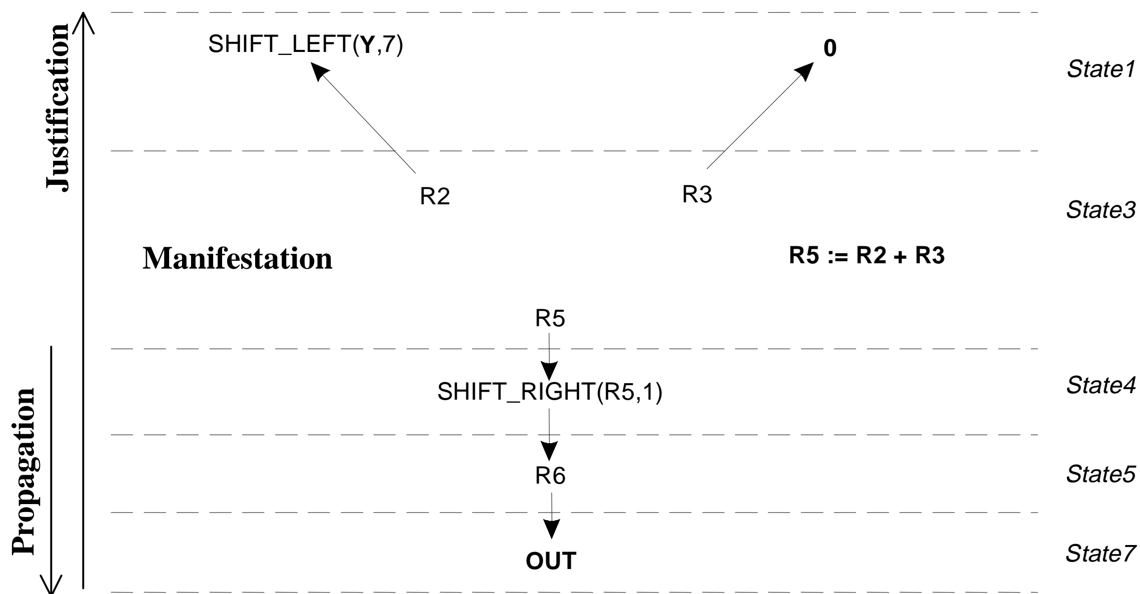


Figure 5.13. The state sequence activated by high-level test

In Figure 5.13, the state sequence activated during high-level test for operation R2+R3 is presented. The propagation and justification stages of this test were shown in Figure 5.9 and Figure 5.10, respectively. Figure 5.14 explains the constraint extraction (i.e. creating and updating) during this sample test.

The final results are shown on top of a gray background. Note that the extracted conditional constraints are inconsistent. According to the first path activation constraint the expression $0 + 1 \leq 6$ must be false, which is obviously not the case. Thus, backtrack will occur and the test generation algorithm will try to activate an alternative high-level test path.

5.3.5 Constraints solving and low-level test

Above we have described the alforithm up to the stage, where the high-level tests path is activated and our task is to satisfy the extracted constraints and test the datapath module (MUT) corresponding to current node under test at the low-level. As it can be seen from the example in Figure 5.14, the extracted constraints are not always satisfiable. They can also be inconsistent or too complex for the constraint satisfaction algorithm to solve. In these cases, a backtrack occurs and the high-level test generation algorithm attempts to activate an alternative test path.

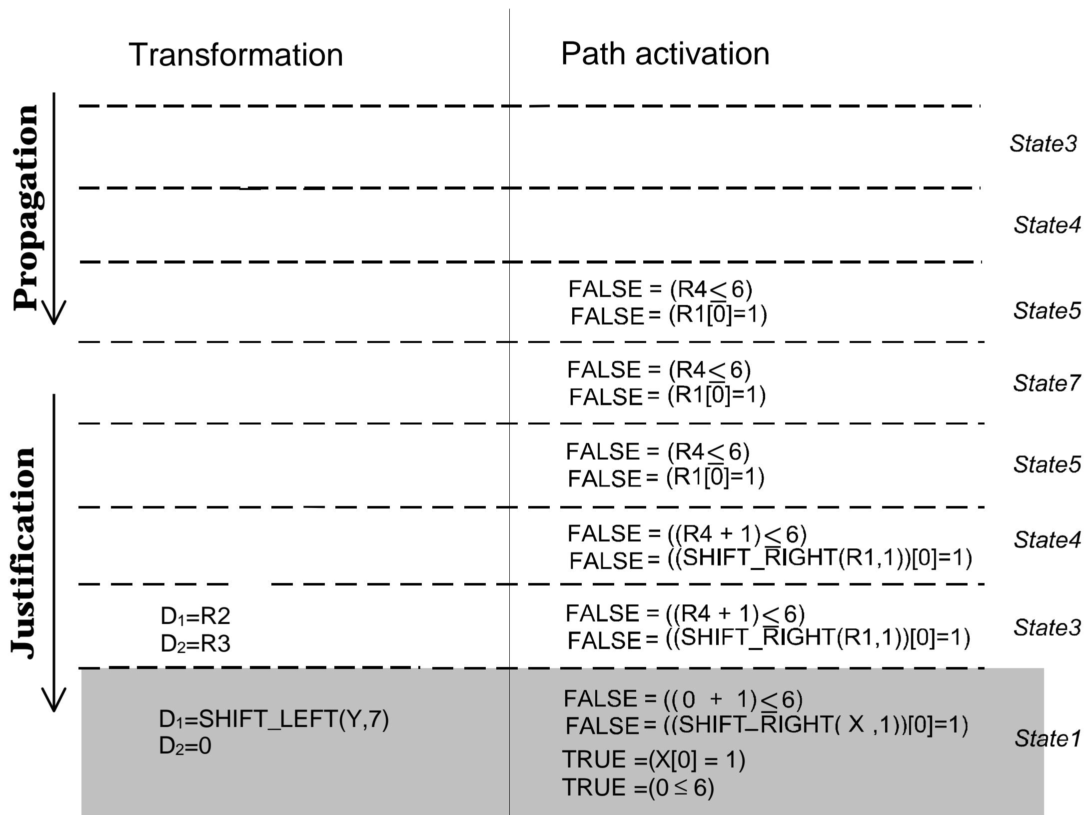


Figure 5.14. Extraction of test generation constraints.

In general case a single activated path is not enough to reach 100 per cent fault efficiency for a functional unit, i.e. test set for a FU can consist of vectors generated during different activated paths, and therefore, different calls to the low-level part. Thus, a record is kept about the faults detected by the low-level tests during previous activated paths. Figure 5.15 shows the data flow of interaction between high-level and low-level parts of the algorithm.

Similar to [FFS98], in present implementation we handle data dependent loops by limiting the number of iterations in these loops. Therefore, the search for high-level paths takes place inside this bound of iterations only. In addition, we use the number of total decisions during the path activation as the timeout limit that terminates the search.

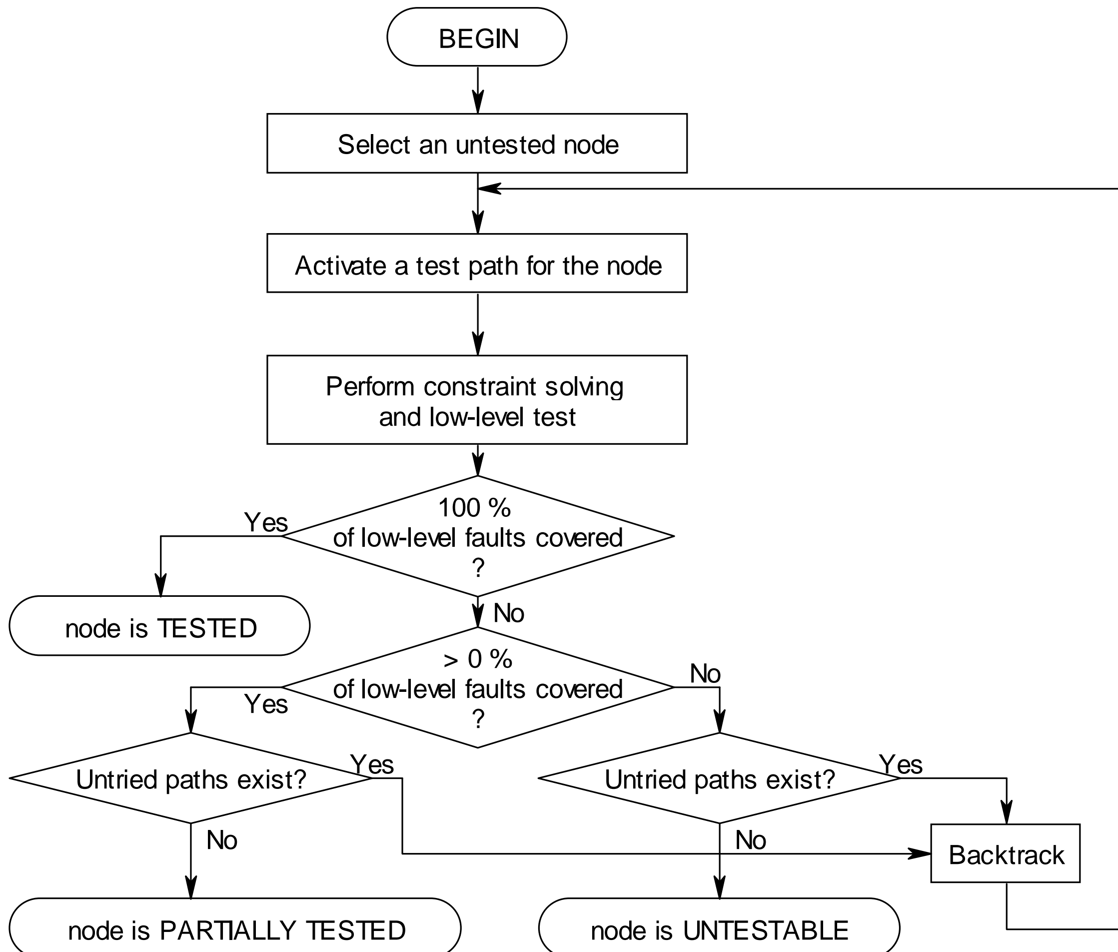


Figure 5.15. Interaction between high- and low-level parts of the algorithm

In current hierarchical approach, constraint solving and low-level test take place as follows. Let $C_i=g_i(x')$, $x' \subseteq x_I$ be current path activation constraints, and $D_j=h_j(x'')$, $x'' \subseteq x_I$ be the transformation constraints, respectively. Constraint satisfaction is applied to the constraints $C_i=g_i(x')$. In current implementation these constraints are solved by generating random values to the set of variables x'. If we fail to satisfy any of constraints $C_i=g_i(x')$ after a certain limit L_1 of values have been generated we make a backtrack in the high-level path activation process and search for an alternative path.

Let us refer to the variables whose values are '*X*' (don't care) as *free variables*. If there exists free variables in x'' after constraint solving of path activation constraints then a set of L_2 random values are generated for each such variable. Subsequently, we simulate the constraints $D_j=h_j(x'')$ obtaining the local test patterns of the inputs of the datapath module under test (MUT). These local test pattern values are applied to fault simulation of the MUT at the gate-level. If L_3 subsequent local test patterns do not increase the achieved fault coverage then fault simulation of local tests is terminated.

If $x'' \cap x' \neq \emptyset$ then we iterate the constraint satisfaction and low-level test procedures L_4 times in order to minimize correlation between constraint satisfaction solutions and local test data. The parameters L_1, ..., L_4 can be set for current implementation at the command line.

5.4 Test Generation Example

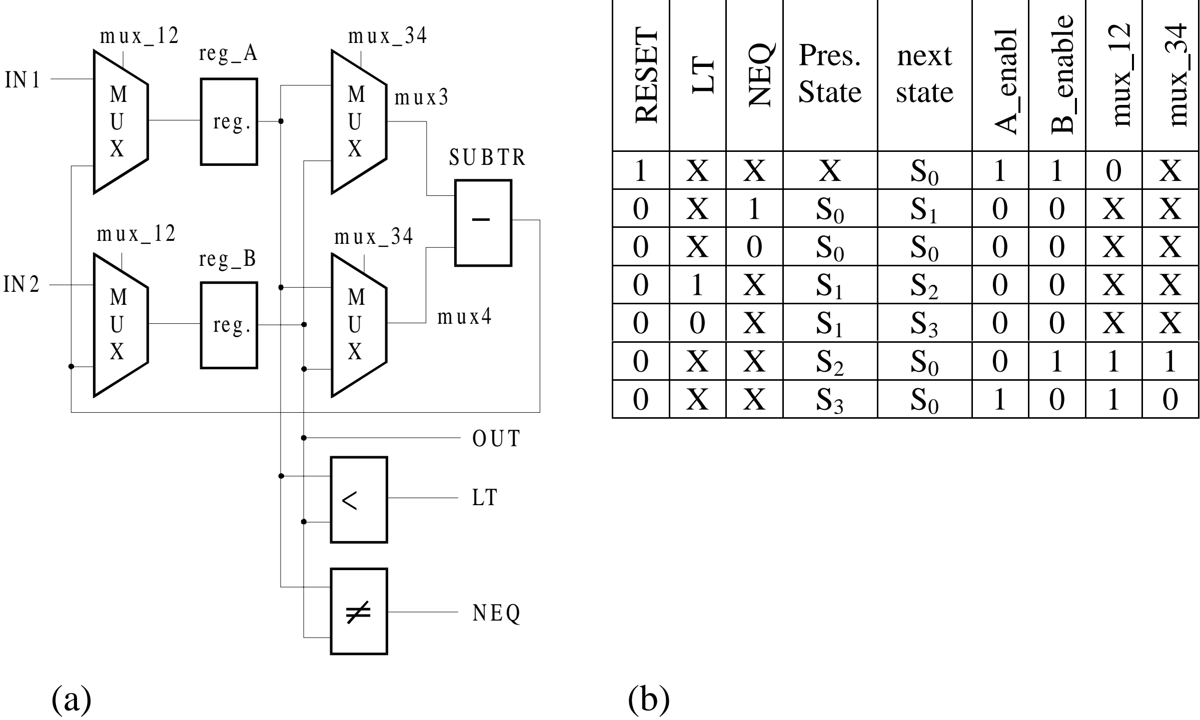
In the following, the hierarchical test generation algorithm is explained basing on a simple example of the Greatest Common Divisor (GCD) circuit. Consider the GCD algorithm described at behavioral level in a pseudo hardware description language presented in Figure 5.16.

Let us assume that subsequent to applying high-level synthesis to the algorithm description in Fig. 5.16, we obtain the RTL architecture presented in Fig. 5.17. This architecture consists of a datapath of 3 Functional Units (FU), 2 registers and 4 multiplexers and a control part Finite State Machine (FSM) of four states. The datapath architectutre is depicted in Figure 5.17a and the control part is given as a state table in Figure 5.17b, respectively.

Decision Diagram (DD) representation for the above architecture is presented in Figure 5.18. The DD model consists of 6 graphs and 28 nodes. For convenience of explaining the example below, terminal nodes of the FSM DD are indexed by numbers in italics. Note, that the dotted lines with arrows that are shown in the figure do not belong to the DD model but are provided simply to make it easier to notice some of the correspondences between variables and respective nodes labeled by them.

```
A := IN1;
B := IN2;
while (A ≠ B)
  if (A < B) then
    B := B – A;
  else
    A := A – B;
  end if;
end while;
OUT := A;
```

Figure 5.16. HDL description of the GCD example



RESET	LT	NEQ	Pres. State	next state	A_enabl	B_enable	mux_12	mux_34
1	X	X	X	S_0	1	1	0	X
0	X	1	S_0	S_1	0	0	X	X
0	X	0	S_0	S_0	0	0	X	X
0	1	X	S_1	S_2	0	0	X	X
0	0	X	S_1	S_3	0	0	X	X
0	X	X	S_2	S_0	0	1	1	1
0	X	X	S_3	S_0	1	0	1	0

Figure 5.17. RT-level architecture of the GCD circuit

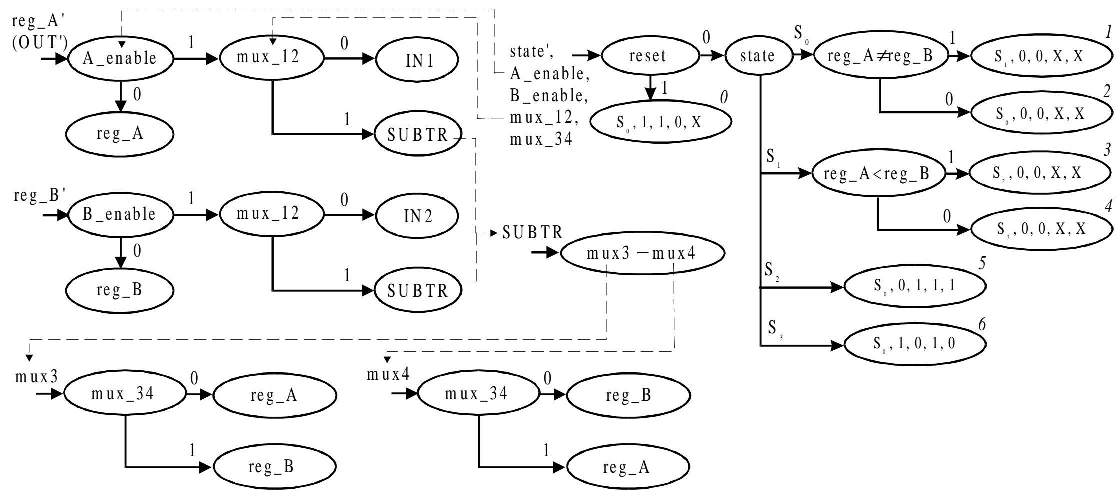


Figure 5.18. DD model of the GCD circuit

We explain the test generation algorithm described in Section 4 by the example of generating scanning test for the terminal node *mux3 – mux4* in the graph calculating the value of *SUBTR*.

Fault manifestation. Set all the variables to 'don't care' values. Create transformation constraints D_0=*mux3*, D_1=*mux4*. Set the fault effect pointer to variable *SUBTR*, i.e. $p_D \rightarrow$ *SUBTR*.

Fault effect propagation. Choose a datapath graph containing a terminal node labeled by *SUBTR*. There are two possible choices: graph for *reg_A'* and graph for *reg_B'*, respectively. Let us select the first choice. Subsequently, we activate the path to the terminal node labeled by *SUBTR* in graph *reg_A'*. This results in following variable assignments: *A_enable := 1, mux_12 := 1*. Since *reg_A'* is a register (*reg_A'* $\in x_R$), and the state variable holds the 'don't care' value ($x_S = X$), we have to choose a consistent FSM terminal node and activate the path to this node in the FSM graph. The only terminal node consistent with previous variable assignments is the terminal node with index 6 (labeled by vector S_0, *1, 0, 1, 0*). Subsequent to activating the path to this node we obtain the following assignments: *reset:=0*, *B_enable := 0*, *mux_34 := 0*, *state* := S_3 (in current clock cycle), *state* := S_0 (in the next clock cycle). We move to the next clock cycle and set the fault effect pointer p_D to *reg_A' (OUT)*.

We detect that the fault effect pointer points to a variable corresponding

to a primary output and successfully completing the fault propagation process.

Constraints justification. As there were no path activation constraints created during manifestation and propagation stages, we move backwards in terms of clock-cycles until the clock-cycle of manifestation phase is reached. We select the justification objective from the unjustified variables of the transformation constraints (D_1=*mux3*, D_2=*mux4*). Let current objective be to justify variable *mux3*. Let us activate the path to the consistent terminal node in graph G_{SUBTR}. Due to the fact that we have already assigned *mux_34 := 0* at current clock-cycle during the propagation process, the consistent terminal node is the one labeled by *reg_A*. Then, we apply update to the constraints, obtaining D_1= *reg_A*, D_2= *reg_B* and move to the preceeding clock cycle.

Without focusing on further details, we continue executing the constraint justification algorithm until the constraints presented in Figure 5.19 is activated as one of possible high-level solutions. In the Figure we have denoted the manifestation clock cycle by *t*, the *i*-th cycle following *t* is denoted by *t+i* and *i*-th cycle preceeding *t* is denoted by *t-i*, respectively. Below the clock-cycle information, the activated state sequence is provided. Then we present graphically the processes of fault propagation and extraction of transformation constraints. Decisions in the high-level path activation are marked by stars (*) in the Figure. Extraction of path activation constraints is depicted below the striped line. Here, *t* corresponds to Boolean value 'true' and *f* corresponds to 'false'. As shown in Figure 5.19, we have to apply the constraint satisfaction process to the following set of constraints: in1 < in2 is false, in1 ≠ in2 is true.

If subsequent to testing the node at the low-level with the first path we have not covered 100 % of stuck-at faults in the FU SUBTR, backtrack occurs and the high-level test generation algorithm tries to find test paths.

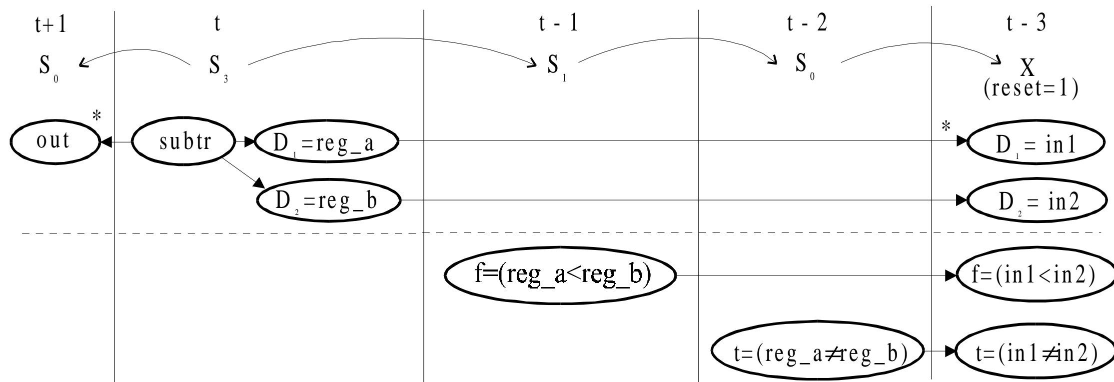


Figure 5.19. High level path activation example

5.5 Experimental Results

In Table 5.1, main characteristics of the benchmark circuits are presented. The following benchmarks were included to the test experiment: Greatest Common Divisor (GCD), Differential Equation (DIFFEQ) and an 8-bit multiplier (MULT8x8). The behavioral descriptions of GCD and DIFFEQ were taken from a benchmark set of [HLS92] and the description of MULT8x8 from [FUT95].

Table 5.2 shows comparison of test generation results of three ATPG tools. These are DECIDER [Rai00] (based on the approach described in current thesis), GATEST [Rud94] and HITEC [Nie91], respectively. GATEST is a genetic algorithm based test generator. HITEC is a deterministic gate-level ATPG. Both of the tools have been developed at the University of Illinois at Urbana-Champaign.

The experiments were run on a 300 MHz SUN UltraSPARC 10 workstation with 128 MB RAM under SOLARIS 2.6 operating system. Actual stuck-at fault coverages of the test patterns generated by all the three tools were measured by the fault simulator from TURBO TESTER system [JMP98], created at Tallinn Technical University, Estonia. As it can be seen from Table 5.2, HITEC offers the poorest performance, both, in terms of fault coverage and test generation time on all the three example circuits. Fault coverages achieved by GATEST and DECIDER

are almost equal. GATEST reaches equal fault coverage for the *gcd* circuit and only slightly higher coverage for *diffeq*. DECIDER in turn has a 2.1 % advantage in the case of *mult8x8* example. However, the test generation times of the two tools differ greatly. DECIDER spends 26 - 615 times less CPU time for test pattern generation process than GATEST.

The main shortcoming of DECIDER is that the tests generated are roughly 6-7 times longer than those of GATEST. This issue is discussed in detail in the next Chapter, where a new fast static compaction technique is proposed in order to minimize sequential circuit tests.

Table 5.1. Benchmark circuits characteristics

Circuit	Gates	Faults	PIs	POs	Flip-flops	FSM states
gcd	227	844	9	4	15	8
mult8x8	1058	3915	17	16	95	8
diffeq	4195	15,836	81	48	115	6

Table 5.2. Test generation results

	DECIDER			GATEST [Rud94]			HITEC [Nie91]		
	fault cov., %	time, s	test length, vec.	fault cov., %	time, s	test length, vec.	fault cov., %	time, s	test length, vec.
gcd	**92.2**	**3.4**	932	**92.2**	89.8	159	89.3	195.6	198
mult8x8	**79.4**	**13.6**	3020	77.3	1585	515	63.5	1793	124
diffeq	96.0	**15.8**	3805	**96.0**	9720	514	95.1	N. A.	700

5.6 Conclusions

Current Chapter describes a new hierarchical test generation method and its implementation based on using Decision Diagrams (DDs). DDs provide for a joint description of structural properties, functions, and faults. Differently from known methods both, control and datapath parts, and also both, higher and lower design abstraction levels are handled by uniform mathematical mechanisms.

A novel concept of mixed-level combination of deterministic and simulation-based techniques in test generation is introduced. On the RT-level, deterministic path activating is combined with simulation-based techniques used in constraints solving. The gate-level local test patterns for components are randomly generated driven by high-level constraints and partial path activation solutions. Fault coverage of test patterns is measured in terms of stuck-at faults. However, since the low-level final test pattern generation is simulation based, the general method is not fault model dependent, and any low-level fault analysis may be used to achieve high fault coverage for desired fault classes like bridging faults, delay faults, opens, or others.

Comparison with the state-of-the-art test generators for sequential circuits HITEC and GATEST show that the proposed approach achieves at least the same fault coverage with a speed 26 – 607 times higher than its competitors. The dramatic increase in speed of test generation was reached by implementing simplified fault propagation along single path, and constraint justification, where transparency rules are neglected. The simplification of fault propagation may, in some cases, lead to a decrease in fault coverage because potential fault masking is not considered. However, our experiments have not so far indicated any decrease in fault coverage caused by fault-masking in propagation.

The disadvantage of the hierarchical approach presented in this thesis is that the number of vectors in generated tests tends to be larger than it is for the simulation-based gate-level test generators (e.g. GATEST). However, this problem can be overcome by applying static compaction to the tests or by implementing high-level fault collapsing. The former is discussed in the following Chapter, while the latter remains a topic of future research.

In addition, the future work is directed towards improving the fault coverage by refining the fault propagation technique, and towards finding more efficient solutions for the subtask of constraint solving by implementing more sophisticated constraint satisfaction methods.

6 Fast Static Compaction of Sequential Circuit Tests

Minimization of the number of patterns in a test set is an essential problem for the chip manufacturer, who faces the test of millions of units per annum. The time required to test a chip by the ATE is directly proportional with the length of the test sequence. Therefore, the number of patterns in a test set is an important parameter when speaking of test pattern generation. Current paper presents a new technique for static compaction of sequential circuit tests that are divided into independent test sequences. The proposed method offers significantly faster performance in terms of run times than earlier methods.

6.1 Introduction

Minimization of the number of patterns in a test set is an essential problem for the chip manufacturer, who faces the test of millions of units per annum [Tho96]. The time required to test a chip by the ATE is directly proportional with the length of the test sequence. Therefore, the number of patterns in a test set is an important parameter when speaking of test pattern generation. Minimization of this number is refered to as *test set compaction*.

There exist two types of test compaction techniques: static and dynamic. In *static compaction* [Pom96a,Cor97,Pom97], a test sequence is generated and subsequently attempts are made to shorten it without reducing its fault coverage. The main advantage of the static techniques is that they are independent of the adopted ATPG tool. *Dynamic test set minimization* [Goe79, Pom96b, Rud97], on the other hand, is performed at the time when tests are being generated. This requires modification of the test generation algorithm itself in order to make it generate shorter sequences.

As it was seen from the experiments presented in previous Chapter the test sets generated by the hierarchical ATPG proposed in this thesis were considerably longer than the ones of its competitors. What does it mean in respect to test set compaction? Does it imply that when we will perform static compaction to all these test sets then the hierarchical compacted tests are still longer than its simulation-based counterparts?

The answer is: not always. Furthermore, our experience shows that in most of the cases it is the other way round. Longer tests include more sampling data for the compaction algorithm to choose from, resulting potentially in a more optimal solution than the test set selected from a restricted number of patterns. The following table illustrates this trend by presenting comparison of test set compaction for commercial gate-level deterministic ATPG tests and high-level ATPG tests.

Table 6.1. Static compaction of high-level ATPG and gate-level ATPG tests

IS	BW	ATPG	Number of tests	Number of compacted tests	Compaction ratio
4	4	high-level	63	**25**	60%
		gate-level	30	**25**	17%
4	8	high-level	63	**29**	54%
		gate-level	45	33	27%
4	16	high-level	63	**29**	54%
		gate-level	63	36	43%
4	32	high-level	63	**29**	54%
		gate-level	77	45	42%
8	4	high-level	120	30	75%
		gate-level	45	**25**	44%
8	8	high-level	120	**30**	75%
		gate-level	52	31	40%
8	16	high-level	120	**29**	76%
		gate-level	61	41	33%
8	32	high-level	120	**30**	75%
		gate-level	75	50	33%
16	4	high-level	224	39	83%
		gate-level	46	**32**	30%
16	8	high-level	224	43	81%
		gate-level	64	**42**	34%
16	16	high-level	224	**42**	81%
		gate-level	73	48	34%
16	32	high-level	224	**42**	81%
		gate-level	84	59	30%

These experiments were run on a family of combinational benchmark ALUs [Uba98b] with different set of supported operations and different bitwidth. In the Table, IS denotes the number of operations that can be performed by the ALU, BW denotes the bitwidth of the ALU. High-level ATPG is a tool implementing a fault model consisting of scanning and conformity tests, similar to the approach presented in this thesis. Gate-level ATPG is a commercial test generator for combinational and full-scan circuits provided by Synopsys. The two following columns show the number of vectors in the test before and after compaction, respectively. (All the test sets considered in this experiment covered 100 % of detectable stuck-at faults). The last column shows the achieved compaction ratio, which represents the percentage of patterns removed from the original test set. The bigger the ratio, the higher the amount of compaction.

As it can be seen from the Table, the compaction ratio of the high-level ATPG tests is always higher, ranging from 54-84 %, while the gate-level tests can be compacted with the ratio of only 17-44 %. While the initial test sets of the high-level ATPG are considerably larger than those of the gate-level ATPG, after compaction the tests are closer to minimum in most of the cases. Thus, static compaction of test sets for the hierarchical approach presented in this thesis could be an efficient solution eliminating the problem of the relatively long tests. In this Chapter we are going to present a new method for static compaction of sequential circuit test sets that is faster than earlier methods, while providing still very good compaction.

Most of the earlier works in the field of static compaction [Pom96a,Pom97] consider the case, where there is a single test sequence that we are trying to minimize by removing some patterns from it. This requires iterative fault simulation during the compaction process in order to check that the fault coverage has not decreased. Thus the run times are very long.

Faster approach has been proposed in [Cor97] and in [Hsi99]. The technique in [5] requires keeping track of the internal state of the circuit. In [Cor97] the whole test set is divided into independent test sequences separated by global reset and fault simulation is performed only once, prior to compaction. In addition, in [Cor97] a set of benchmarks [CAD] consisting of 103 fault matrices of ISCAS89 circuits tested by three

different ATPG tools [Cor96, Nie91, Cab93] were made publicly available.

In this Chapter we consider the above mentioned case of static compaction, where the test set is divided into test sequences. Note, that this case is particularly suitable for current hierarchical test generation approach, because each high-level test can be viewed as an independent test sequence. We propose a new technique, which allows better compaction than any previously published method belonging to this class. Moreover, the technique is about two orders of magnitude faster than the genetic approach to solving the same problem published in [Cor97]. The proposed technique allows to calculate lower bounds for the compaction problem and therefore is capable of achieving and proving globally optimal results for most of the benchmark test sets.

The technique uses an effective representation of fault matrices by weighted bipartite graph models which provide for a more compact means of describing the test sets than traditional matrix representations. It contains a preprocessing step for determining the set of essential vectors of the test sequences. This step considerably reduces the search space for the compaction algorithm. Subsequent to preprocessing, a greedy search algorithm is applied in order to compact the test set.

The Chapter is organized as follows. In Subsection 6.2 the task of static compaction of test sets consisting of multiple sequences is explained. In addition, we will show how traditional fault matrices can be converted into weighted bipartite graphs. Subsection 6.3 explains the compaction algorithm. In Subsection 6.4, detection of lower bounds and proving globally optimal results is defined. Subsection 6.5 presents the experimental results and finally conclusions are given.

6.2 Basic Concepts and Model Representation

In this Subsection we explain how a test set consisting of separate test sequences can be represented by fault matrixes and bipartite graphs. The problem definition of static compaction of the test sets is presented.

A *test set* T consists of *test sequences* $t_i \in T$, $i = 1, \ldots, n$. Each sequence t_i contains in turn L_i *test vectors*. We refer to L_i as the *test length* of sequence t_i. The set of faults f_j, $j = 1, \ldots, m$ detected by T is denoted by F. Total test

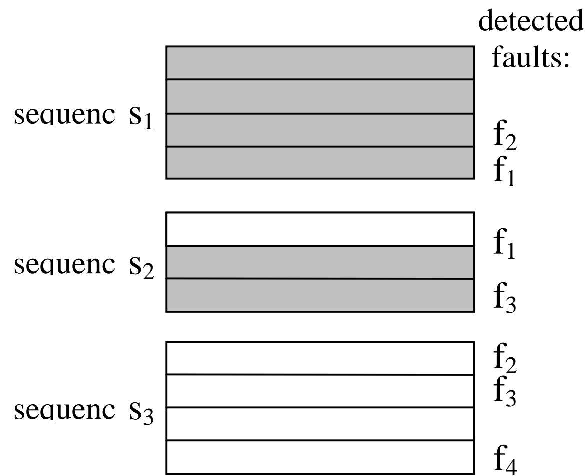


Figure 6.1. Test set compaction example

length of test set T can be viewed as a sum

$$\sum_{i=1}^{n} L_i .$$

In the problem of static compaction of test sequences, our task is to find values for the L_i such that the above sum would be minimal while the number of faults that the test set T detects would still be $|F|$.

Consider the test set example shown in Fig. 6.1 that consists of three test sequences s_1, s_2 and s_3, respectively. Sequence s_1 consists of four test vectors covering fault f_2 at the third vector and f_1 at the fourth vector. Sequence s_2 consists of three test vectors covering f_1 at the first vector and f_3 at the third vector. Finally, sequence s_3 consists of four test vectors covering f_2 at the first vector, f_3 at the second vector and f_4 at the fourth vector.

Initial test length of this test set is 11 vectors. It can be found that the optimal solution for the static compaction problem is selecting sequence s_3 and the first vector from sequence s_2. Hence, the length of the optimal compacted test set will be 5 vectors.

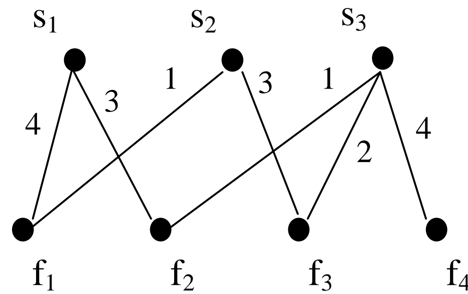


Figure 6.2. Weighted bipartite graph for the test set in Fig. 6.1

It is possible to represent the test set by a matrix, where the rows correspond to sequences and columns correspond to faults [Cor97]. This representation for the test set in Fig. 6.1 is shown in Table 6.2.

Table 6.2. Fault matrix for the test set of Fig. 6.1

	f_1	f_2	f_3	f_4
s_1	4	3	0	0
s_2	1	0	3	0
s_3	0	1	2	4

In this type of descriptions test set T consisting of n faults and m test sequences can be viewed as a matrix

$$T = \begin{pmatrix} t_{f_1,s_1} & t_{f_2,s_1} & \cdots & t_{f_n,s_1} \\ t_{f_1,s_2} & t_{f_2,s_2} & \cdots & t_{f_n,s_2} \\ \cdots & \cdots & \cdots & \cdots \\ t_{f_1,s_m} & t_{f_2,s_m} & \cdots & t_{f_n,s_m} \end{pmatrix},$$

where $t_{si,fj}$ is equal to k if sequence s_i covers fault f_j at the k-th vector and zero if sequence s_i does not cover fault f_j.

If we select k vectors from sequence s_i then all the faults $\{f_j : k \geq t_{si,fj} > 0\}$ are said to be covered by these vectors. In our algorithm we remove the

columns corresponding to the covered faults from matrix *T*. In addition, we must subtract *k* from all the non-zero elements $t_{si,fj}$ of the row corresponding to the sequence s_i.

Our task is to cover all the faults (i.e. columns of matrix *T*) by selecting the minimal number of vectors. As it was shown in [Cor97], this task belongs to the class of NP-complete problems.

In current implementation the test set information is represented by a model of weighted bipartite graphs. The motivation for this is the fact that bipartite graph models generally provide for a much more compact means of describing the test sets than matrix representations. We use a *weighted bipartite graph* $G_{n,m}$, where the first part of the graph consists of n vertices that correspond to test sequences s_i and the remaining part has m vertices corresponding to the faults f_j detected by the test set. There exists an edge connecting vertices s_i and f_j iff sequence s_i covers fault f_j. Edge $e = <s_i, f_j>$ is labeled by an integer c, $c = w(e)$, where fault f_j is covered at the c-th vector of test sequence s_i. The weighted bipartite graph representation for the test set in Fig. 6.1 is shown in Fig. 6.2.

6.3 Compaction Algorithm

A simple pre-processing step of detecting *essential vectors* from the test sequences is applied at the beginning of the compaction algorithm. If fault f_j is detected by the *k*-th vector of test sequence s_i and is not detected by any other sequence then *k* first vectors of sequence s_i are called essential. After selecting the essential vectors we remove them from the test sequences. In addition we remove the columns corresponding to faults covered by these vectors from matrix *T*. This simple pre-processing step allows to significantly reduce the search space for the static compaction algorithm.

In addition to selecting the set of essential patterns, two other types of steps to prune the search space are performed. These are collapsing of equivalent faults and removing dominating sequences, respectively. During *collapsing of equivalent faults*, column f_a will be removed from matrix *T* if there exists another column f_b, where

$$\forall_{i=1}^{m} t_{f_a,s_i} = t_{f_b,s_i}.$$

In other words, if we have multiple identical columns we will unite them into a single one.

Another type of implications is *removing dominating sequences*. A row corresponding to sequence s_b is said to be a dominating sequence of s_a iff

$$\forall_{j=1}^{n} t_{s_b,f_j} \neq 0 \Rightarrow t_{s_a,f_j} \neq 0,\ t_{s_a,f_j} \leq t_{s_b,f_j}.$$

Current technique applies above described implications as far as possible. When it encounters a selection between alternative solutions, it switches to a greedy algorithm [Edm71]. In order to solve a problem, a greedy algorithm proceeds step-by-step, looking for a next solution that would constitute the optimization of the objective function. At each step, the algorithm makes a choice between the candidate solutions basing on a *selection function*. Once a candidate is included in the solution it is never discarded. Greedy algorithm stops when a final solution for the optimization problem is found. It always finds a solution for the optimization problem but it cannot guarantee whether the solution is the global optimum of the objective function. The efficiency of greedy algorithms depends on how well the selection function has been chosen in order to cope with the type of the optimization task.

The greedy selection function implemented in current technique is described in the following. Let us denote by *Minrange*(f_j) the minimal number of vectors that has to be selected from any test sequence in order to detect a fault f_j. Let *Maxrange* be the maximum *Minrange*(f_j) of all the faults.

The selection function selects *Maxrange* vectors from the corresponding test sequence. If there exist multiple maximal *Minrange*(f_j) values then the algorithm prefers the sequences that detect more faults in *Maxrange* first vectors.

```
Select essential vectors.
Remove the faults covered by these vectors.
While exist uncovered faults
{
   Remove dominating sequences.
   Collaps equivalent faults.
   If new essential vectors appeared then
      Select essential vectors.
   Else
      Select vectors by greedy selection.
   Endif
   Remove the faults covered by selected vectors.
}
```

Figure 6.3. Static compaction algorithm

In Figure 6.3 the description of the algorithm for static compaction is presented.

6.4 Detecting Lower Bounds and Global Optima

Since the algorithm described in the previous section is using implications, it allows it to calculate the lower bounds for the static compaction task. The meaning of the lower bounds is that they show that it is not possible to compact the test set to contain fewer vectors than the bound. Moreover, in the cases where the result found by the algorithm equals the lower bound we have proved that this result is the global optimum.

In current approach, the lower bound is determined by our technique with the number of vectors selected during the implications up to the first greedy selection, including the vectors chosen by that selection. The first greedy selection can be included due to the fact that, obviously, it represents the minimal number of vectors that are necessary in order to cover a previously uncovered fault f_j. All the alternative combinations of selecting vectors for covering f_j must always result in a greater or equal number of vectors.

The fact that the first greedy selection is included to the lower bound does not imply that the globally optimal result must (or even may) contain this selection. As it was mentioned above, greedy algorithms cannot guarantee globally optimal results and there is no reason why the first greedy selection has to be any more correct than the following ones.

Experiments show that the proposed technique achieves compacted test sets that are in most cases equal to the detected lower bounds.

6.5 Experimental Results

Both, the experiments of current approach and the comparative experiments of [Cor97] were run on SUN SPARC 5 computer. We used the test set benchmarks that can be downloaded from [CAD]. The benchmarks include test sets for three different ATPG tools: GATTO [Cor96], HITEC [Nie91] and SYMBAT [Cab93]. GATTO is a genetic algorithm based ATPG, HITEC is a deterministic gate-level ATPG and SYMBAT is based on symbolic test generation techniques. The compaction results for these benchmarks are presented in Tables 6.5, 6.6 and 6.7, respectively.

Experiments show that current technique offers 16,7 - 294 times shorter CPU times and is in average 74,3 times faster than the method implemented in [Cor97]. The run time statistics for test set benchmarks of different ATPG tools are presented in Table 6.3.

Table 6.3. Speed-up in comparison to [Cor97].

Speed-up, times	Average	Maximum	Minimum
GATTO test sets	77.1	218.2	16.7
HITEC test sets	83.1	294.1	20.0
SYMBAT test sets	54.3	200.0	22.0

Table 6.4 and Figure 6.4 show the average compaction achieved by current approach in comparison to compaction of [Cor97] and the theoretical lower bound calculated by current technique. As it can be seen from the Table, compaction of this technique is in average better than the one of [Cor97] for all the ATPG test sets. In the case of HITEC and

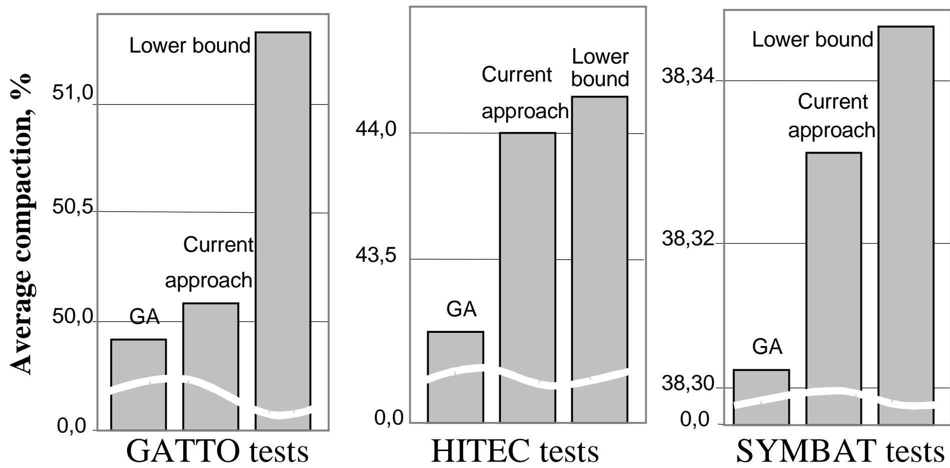


Figure 6.4. Comparison of average compaction

SYMBAT the technique allows much closer to optimum results than reported in [Cor97].

Table 6.4. Average compaction of test sequences.

	GA [Cor97]	Current approach	Lower bound
GATTO test sets	49.86 %	50.06 %	51.28 %
HITEC test sets	43.30 %	43.98 %	44.15 %
SYMBAT test sets	38.30 %	38.33 %	38.35 %

Tables 6.5-6.7 present comparative experimental results. The best compaction result for a benchmark test set is denoted by bold digits. In cases, where globally optimal result was proved, the number is followed by an asterisk character. Current technique is capable of detecting globally optimal results for 50 % of the GATTO test sets, 82.5 % of HITEC test sets and 91.3 % SYMBAT test sets, respectively. This is considerably more than any of the previously published works on these benchmarks.

6.6 Conclusions

In this Chapter, the case of static compaction where the test set is divided into test sequences is considered. A new technique is proposed that allows better compaction than any previously published method belonging to this particular class. Moreover, experiments show that the technique offers 16.7-294 times shorter CPU times and is in average 74.3 times faster than the method presented in [Cor97].

Unlike previously published approaches, this technique is capable of detecting and proving globally optimal results for most of the benchmark test sets in [CAD]. Global optima are proved for 50 % of GATTO test sets, 82.5 % of HITEC test sets and 91.3 % of SYMBAT test sets using this approach.

As a future work, it is possible to further improve the technique by implementing an exact algorithm basing on the branch-and-bound approach.

Table 6.5. Compaction results for GATTO test sets

circuit	Initial test set		Compaction test in [Cor97]			Essential test		Lower bound	Compacted test set		
	# seq.	# vec.	# seq.	# vec.	time, s	# seq.	# vec.	# vectors	# seq.	# vec.	time, s
s208	36	1096	6	**347**	8,00	4	290	347	6	**347***	0,08
s298	24	302	11	**141**	8,00	10	130	141	11	**141***	0,09
s344	19	141	10	**66**	7,00	6	49	63	10	**66**	0,08
s349	19	144	11	**84**	7,00	9	75	81	11	**84**	0,09
s382	17	840	7	**485**	8,00	7	485	485	7	**485***	0,08
s386	38	418	15	**221**	17,00	11	199	221	15	**221***	0,11
s400	16	916	7	**502**	9,00	6	443	502	7	**502***	0,06
s420	33	797	8	**333**	21,00	5	325	329	8	**333**	0,10
s444	22	1434	9	**788**	12,00	9	788	788	9	**788***	0,11
s499	29	465	9	**192**	19,00	8	182	192	9	**192***	0,19
s510	37	989	7	**237**	18,00	4	146	210	7	263	0,32
s526	18	1050	9	**769**	14,00	8	696	769	9	**769***	0,09
s526n	16	862	6	**523**	12,00	6	523	523	6	**523***	0,08
s641	48	395	24	**221**	25,00	23	219	221	24	**221***	0,24
s713	55	557	23	**250**	10,00	19	226	250	23	**250***	0,37
s820	38	669	14	**347**	14,00	13	335	346	14	**347**	0,19
s832	10	425	10	**196**	12,00	10	184	196	10	**196***	0,16
s838	37	1323	11	**473**	16,00	5	323	412	11	482	0,18
s938	37	1323	11	**473**	16,00	5	323	412	11	482	0,18
s953	75	1099	32	**539**	20,00	26	447	538	32	**539**	0,91
s967	72	1223	31	**669**	19,00	27	606	665	32	679	0,82
s991	20	448	9	**365**	5,00	9	365	365	9	**365***	0,30
s1196	133	1805	73	1131	43,00	67	1086	1124	74	**1124***	0,93
s1238	123	1554	72	1007	50,00	62	956	976	74	**1004**	0,91
s1269	52	450	29	**245**	17,00	23	198	217	29	**245**	0,70
s1423	107	2691	28	1284	46,00	25	1237	1279	28	**1279***	0,99
s1488	65	1824	19	**946**	24,00	16	867	946	19	**946***	1,07
s1494	62	1244	19	**652**	25,00	19	652	652	19	**652***	0,89
s1512	52	772	14	**289**	27,00	12	261	287	15	294	0,55
s3271	132	2529	50	1532	111,00	43	1047	1087	53	**1210**	5,40
s3330	108	2028	44	**1067**	110,00	39	1018	1061	45	1070	2,86
s3384	58	888	22	**410**	60,00	17	327	386	22	**410**	2,85
s4863	112	1533	42	**746**	154,00	31	666	721	42	749	5,64
s5378	71	919	41	**493**	139,00	37	464	472	42	**493**	3,24
s6669	64	592	36	303	114,00	29	240	291	36	**301**	6,62
s13207	34	544	9	**187**	207,00	6	159	187	9	**187***	1,7
s15850	10	153	3	**91**	96,00	2	79	91	3	**91***	0,44
s35932	59	903	8	**308**	706,00	6	247	308	8	**308***	25,00
s38417	95	1617	31	**697**	1601,00	22	588	651	31	698	11,78
s38584	271	8065	108	N/A	4918,00	95	3416	3705	106	**3812**	60,00

* - **the compaction technique proved the globally optimal result.**

Table 6.6. Compaction results for SYMBAT test sets

circuit	Initial test set		Compacted test in [Cor97]			Essential test		Lower bound	Compacted test set		
	# seq.	# vec.	# seq.	# vec.	time, s	# seq.	# vec.	# vectors	# seq.	# vec.	time, s
s208	64	2049	34	**1354**	4,00	22	1199	1354	34	**1354***	0,05
s298	34	344	17	**193**	3,00	17	193	193	17	**193***	0,11
s344	47	187	23	**111**	4,00	23	111	111	23	**111***	0,15
s349	46	184	24	**113**	5,00	23	111	113	24	**113***	0,15
s382	59	2580	25	**1318**	8,00	25	1318	1318	25	**1318***	0,21
s386	76	340	43	**204**	11,00	39	197	204	43	**204***	0,19
s400	59	2538	26	**1352**	8,00	25	1319	1352	26	**1352***	0,23
s420	49	9377	25	**8748**	12,00	22	8742	8748	25	**8748***	0,06
s444	39	2034	24	**1262**	6,00	23	1228	1262	24	**1262***	0,20
s499	33	418	23	**298**	7,00	23	298	298	23	**298***	0,20
s510	59	1066	41	**810**	9,00	41	810	810	41	**810***	0,41
s526	74	3607	31	**1679**	14,00	30	1636	1679	31	**1679***	0,37
s526n	73	3573	31	**1679**	14,00	30	1636	1679	31	**1679***	0,34
s641	160	516	97	**340**	27,00	96	338	340	97	**340***	0,38
s713	164	538	100	**356**	36,00	96	348	356	100	**356***	0,41
s820	202	1425	109	780	54,00	108	777	777	108	**777***	0,99
s832	195	1370	107	**768**	55,00	107	768	768	107	**768***	0,98
s953	155	1261	85	**761**	46,00	79	710	761	85	**761***	1,41
s967	162	1322	88	**795**	47,00	84	764	795	88	**795***	1,39
s1196	297	613	200	376	109,00	195	365	374	199	**375**	1,27
s1238	300	619	206	385	132,00	199	373	383	204	**383***	1,39
s1488	157	1709	99	**1110**	64,00	94	1099	1110	98	**1110***	1,95
s1494	160	1787	100	**1140**	68,00	94	1118	1136	99	**1140**	1,98

* - **the compaction technique proved the globally optimal result.**

Table 6.7. Compaction results for HITEC test sets

circuit	Initial test set		Compacted test in [Cor97]			Essential test		Lower bound	Compacted test set		
	# seq.	# vec.	# seq.	# vec.	time, s	# seq.	# vec.	# vectors	# seq.	# vec.	time, s
s208	44	741	10	**291**	3,00	6	156	291	10	**291***	0,07
s298	19	217	7	118	2,00	7	116	116	7	**116***	0,05
s344	10	61	6	46	1,00	6	45	45	6	**45***	0,05
s349	15	84	9	64	2,00	9	62	63	9	**63***	0,05
s382	15	359	2	156	3,00	2	155	155	2	**155***	0,04
s386	58	258	31	**162**	8,00	31	161	162	31	**162***	0,14
s400	15	357	2	156	4,00	2	155	155	2	**155***	0,04
s420	51	788	10	275	10,00	9	240	274	10	**274***	0,12
s444	17	308	2	205	4,00	2	204	204	2	**204***	0,04
s510	37	847	27	624	6,00	26	586	623	27	**623***	0,29
s526	17	260	2	173	5,00	2	172	172	2	**172***	0,05
s526n	16	256	2	**169**	4,00	2	169	169	2	**169***	0,05
s641	78	306	36	**170**	12,00	31	150	164	36	**170**	0,30
s713	74	270	34	171	14,00	29	157	168	34	**168***	0,29
s820	120	1170	64	**672**	33,00	60	651	671	62	**671***	0,68
s832	111	1058	60	619	33,00	57	591	617	60	**617***	0,57
s838	52	675	12	**310**	24,00	7	297	306	12	**310**	0,23
s938	52	675	12	**310**	24,00	7	297	306	12	**310**	0,23
s953	111	825	38	**404**	35,00	35	370	404	38	**404***	0,89
s967	120	831	38	**407**	39,00	35	375	407	38	**407***	0,85
s991	50	83	25	**46**	14,00	25	46	46	25	**46***	0,20
s1196	189	509	109	339	69,00	105	322	336	109	**337**	0,82
s1238	191	513	108	**332**	83,00	105	320	332	108	**332***	0,82
s1269	66	255	25	156	28,00	25	131	136	26	**136***	0,44
s1423	49	283	16	188	33,00	15	183	187	16	**187***	0,27
s1488	24	69	16	**56**	18,00	15	54	56	16	**56***	0,17
s1494	60	523	43	424	33,00	40	387	418	43	**418***	0,72
s1512	59	283	14	118	35,00	13	109	117	14	**117***	0,33
s3271	61	984	22	**489**	57,00	12	327	483	19	**489**	2,09
s3330	132	764	85	549	150,00	82	531	548	86	**548***	2,52
s3384	17	212	8	165	22,00	8	163	164	8	**164***	0,59
s4863	105	376	57	257	169,00	55	250	252	57	**256**	2,31
s5378	95	250	49	**152**	194,00	48	151	152	49	**152***	2,65
s6669	68	466	22	**259**	117,00	22	249	259	23	**259***	4,70
s9234	6	19	1	10	50,00	2	9	9	2	**9***	0,17
s13207	14	97	6	58	96,00	6	57	57	6	**57***	1,70
s15850	14	39	4	15	126,00	2	12	14	4	**14***	0,44
s35932	376	1712	13	244	4259,00	6	188	213	11	**242**	122,50
s38417	280	806	14	132	4828,00	11	122	131	14	**131***	21,50
s38584	48	509	30	**435**	1000,00	30	435	435	30	**435***	5,33

* - the compaction technique proved the globally optimal result.

7 Thesis Summary

The topic of this thesis is hierarchical test pattern generation based on multi-level Decision Diagram (DD) models. In Chapter 2, the basic concept of decision diagram representations is given and representing hierarchical designs by means of multi-level DD models was explained. In Chapter 3, simulation on high-level DD representations was discussed. Chapter 4 considered fault modeling at logic level and multi-level DD models. In Chapter 5, a hierarchical ATPG based on multi-level DD models was presented. Chapter 6 explained how it is possible to further optimize the generated test sets by applying fast static compaction methods.

This Chapter summarizes the conclusions of the sub-tasks presented in the thesis and outlines suggestions to improve the presented prototype tool and methodologies.

7.1 Conclusions

In the following, the main scientific results obtained in this thesis are listed. The results are grouped according to the Chapters (sub-tasks) of the thesis.

<u>Multi-level decision diagram representations (Chapter 2)</u>

- The concept of general decision diagrams was presented. Up to the present moment, no commonly accepted concept and terminology for general decision diagrams is available in the literature.

- Multi-level DD representations used for hierarchical fault modeling in current thesis were introduced.

<u>Simulation techniques on decision diagrams (Chapter 3)</u>

- Cycle-based simulation on decision diagram models was investigated. The proposed DD-based event-driven simulator is significantly faster than commercial state-of-the-art cycle-based and event-driven HDL simulators. The speed-up in simulation time is in average 5.6 times compared to cycle-based HDL simulation and 20.1 times compared to event-driven HDL simulation, respectively.

Fault modeling using multi-level decision diagrams (Chapter 4)

- A hierarchical fault model was proposed, where using a combination of dedicated tests for the terminal and non-terminal nodes, 100 % stuck-at coverage for the circuit datapath is achieved.

- Fault collapsing achieved by SSBDD model generation was discussed. The advantage of the SSBDD based collapsing over the traditional one is that it allows us at the same time to rise to a higher abstraction level of circuit modeling. This topic has not been studied in previous works.

- Advantages of using SSBDD representations instead of traditional BDDs in fault simulation were pointed out.

- It was shown that SSBDDs are a feasible circuit model for both, two- and three-valued parallel fault simulation.

Hierarchical test generation on multi-level decision diagrams (Chapter 5)

- A new hierarchical test generation method was proposed, where differently from known methods, both, control and datapath parts, and also both, higher and lower design abstraction levels are handled by uniform mathematical basis.

- Comparison with the state-of-the-art test generators for sequential circuits HITEC and GATEST show that the proposed approach achieves at least the same fault coverage with a speed 26 – 607 times higher than its competitors. The dramatic increase in speed of test generation was reached by implementing simplified fault propagation along single path, and constraint justification, where transparency rules have been neglected.

Fast static compaction of sequential circuit tests (Chapter 6)

- In this thesis, the case of static compaction, where the test set is divided into test sequences is considered. A new technique is proposed that allows better compaction than any previously published method belonging to this particular class. Moreover, experiments show that the technique is in average 74.3 times faster than the method presented in

[Cor97]. Unlike previously published approaches, this technique is capable of detecting and proving globally optimal results for most of the benchmark test sets in [CAD].

7.2 Future Work

The following future improvements to the approach presented in this thesis could be made.

1) In Chapter 4, a hierarchical fault model was proposed, where using a combination of dedicated tests for DD nodes, 100 % stuck-at coverage for the circuit datapath is achieved. As a future work, the fault model could be extended to include all the nodes of control part as well.

2) The test generation algorithm presented in Chapter 5 can be further improved by refining the fault propagation towards higher accuracy.

3) More efficient solutions for the subtask of constraint solving (presented in Chapter 5) should be found by implementing deterministic constraint satisfaction methods.

4) One of our future plans is to implement alternative search strategies during the high-level test path activation presented in Chapter 5. The strategies could include depth-first search (implemented in this thesis), breadth-first search, backjumping, or a combination of the above. This would increase the robustness of the approach by allowing to handle circuits with complicated control flow more efficiently.

5) Finally, it is possible to further improve the static compaction technique presented in Chapter 6 by implementing an exact algorithm basing on the branch-and-bound approach. The technique should be included to the hierarchical test generation system and compaction experiments with the generated tests should be carried out.

References

[Ake78] S.B.Akers, “Binary Decision Diagrams”, *IEEE Trans. on Computers*, Vol. 27, pp.509-516, 1978.

[AM95] P. Ashar, S. Malik: “Fast Functional Simulation Using Branching Programs”, ICCAD Conf., 1995, pp 408-412

[BCMD90] J. R. Burch, E.M. Clarke, K. L. McMillan, D. L. Dill, “Sequential circuit verification using symbolic model checking”, *Proc. 27th ACM/IEEEDAC*, pp.46-51, June 1990.

[Bhat94] S. Bhatia, N.K. Jha, “Genesis: A Behavioral Synthesis for Hierarchical Testability”, Proc. of the European Design and Test Conference, pp. 272-276, 1994.

[Brah84] D. Brahme, J. A. Abraham, "Functional Testing of Microprocessors", *IEEE Trans. Comput.*, vol. C-33, pp. 475-485, 1984.

[Bry86] R. E. Bryant, “Graph-based algorithms for Boolean function manipulation”, *IEEE Trans. on Computers*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

[Bry95] R. E. Bryant, Y.-A. Chen, “Verification of arithmetic functions with binary moment diagrams”, *Proc. 32nd ACM/IEEE DAC*, June 1995.

[Cab93] G. Cabodi, F. Corno, P. Prinetto, M. Sonza Reorda, "Symbat’s user guide", Politecnico di Torino, Sept. 1993.

[CAD] URL: http://www.cad.polito.it

[Chap74] S. G. Chapell, "Automatic test generation for asynchronous digital circuits", *Bell Syst. Tech. J.*, vol. 53, pp. 1477-1503, Oct.1974.

[Che90] K.-T. Cheng, J.-Y. Jou, "A single-state-transition fault model for sequential machines", *Proc. ICCAD’90*, pp. 226-229, 1990.

[Cla93] E. M. Clarke, K. L. McMillan, X. Zhao, M. Fujita, J. Yang, “Spectral transforms for large Boolean functions with applications to technology mapping”, *Proc. 30th ACM/IEEE DAC*, pp. 54-60, June 1993.

[CMF93] O. Coudert, J. C. Madre, H. Fraisse, “A new viewpoint of two-level logic optimization”, *Proc. 30th ACM/IEEE DAC*, pp. 625-630, June 1993.

[Cor96] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits ", *IEEE Trans. CAD*, Aug. 1996.

[Cor97] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, "New static compaction techniques of test sequences for sequential circuits*". Proc.*

ED&TC, 1997, pp.37-43.

[DEI99] K.H.Diener, G.Elst, E.Ivask, J.Raik, R.Ubar: «FPGA Design Flow with Automated Test Generation», Proc. of the 11th Workshop on Test Technology and Reliability of Circuits and Systems, pp. 120-123, 1999

[Dre94] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, M. Perkowski, "Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams", *Proc. 31st ACM/IEEE DAC*, pp. 415-419, June 1994.

[Edm71] J. Edmonds, "Matroids and the greedy algorithm". Mathematical Programming, Vol. 1, 1971, pp. 127-136.

[ETW98] J. Raik, R. Ubar, "Feasibility of Structurally Synthesized BDD Models for Test Generation," *Compendium of Papers of the European Test Workshop*, Barcelona, May 27-29, 1998, pp. 145-146.

[FFK88] M. Fujita, H. Fujisawa, N. Kawato, "Evaluation and improvement of Boolean comparison method based on binary decision diagrams*", Proc. IEEE/ACM ICCAD*, pp. 2-5, Nov. 1988.

[FFS98] F. Ferrandi, F. Fummi, D. Sciuto, "Implicit Test Generation for Behavioral VHDL Models," *Proc. Int. Test Conf.*, pp. 587-596, 1998.

[Flot95] M.L. Flottes, D. Hammad, B. Rouzeyre, "High-Level Synthesis for Easy Testability", *Proc. of the European Design and Test Conference*, pp. 198-206, March 1995.

[FUT95] E. Gramatova, M. Gulbins, M. Marzouki, A. Pataricza, R. Sheinauskas, R. Ubar, "FUTEG Benchmarks," Technical Report of project COPERNICUS JEP 9624 FUTEG No9/1995.

[Gajs89] D.Gajski, N. Dutt, A. Wu, S. Lin, "High-Level Synthesis, Introduction to Chip and System Design", Kluwer Academic Publishers, 1989.

[GF00]I. Ghosh, M. Fujita, "Automatic Test Pattern Generation for Functional RTL Circuits Using Assignment Decision Diagrams", *Proc. 37th ACM/IEEE DAC*, pp. 43-48, June 2000.

[Goe79] P. Goel, B. C. Rosales, "Test generation and dynamic compaction of tests". *Digest of Papers Test Conf.*, 1979.

[Grue98] H. Gruenbacher, M. Khosravipour, G. Gridling: "Improving Simulation Efficiency by Hierarchical Abstraction Transformations", System Level Design Language Workshop, Lausanne, Switzerland, September 1998.

[Gup85] A. Gupta, J. R. Armstrong, "Functional fault modeling", *30th ACM/IEEE DAC*, pp. 720-726, 1985.

[HLS92] HLSynth92 benchmark directory at

URL: http: //www.cbl.ncsu.edu/pub/Benchmark_dirs/HLSynth92/

[Hsi97] M. S. Hiao, E. M. Rudnick, J. H. Patel, "Sequential circuit test generation using dynamic state traversal", *Proc. European Design and Test Conf.*, pp. 22-28, 1997.

[Hsi99]M.S. Hsiao et al., "Fast static compaction algorithms ...". *IEEE Trans. Comp.*, Vol. 48, No. 3, 1999.

[ISC00] R.Ubar, A.Morawiec, J.Raik. Back-Tracing and Event-Driven Techniques in High-Level Simulation with Decision Diagrams, *Proc. of the IEEE ISCAS'2000 Conference*, Vol. 1, pp. 208-211, May 28-31, Geneva, Switzerland.

[JMP98] G.Jervan, A.Markus, P.Paomets, J.Raik, R.Ubar, "Turbo Tester: A CAD System for Teaching Digital Test," Microelectronics Education, Kluwer Academic Publishers, Dordrecht/Boston/London, pp.287-290, 1998.

[Jut00] A. Jutman, R. Ubar, "Design Error Diagnosis in Digital Circuits with Stuck-at Fault Model," *Journal of Microelectronics Reliability. Pergamon Press, Vol. 40, No 2*, 2000, pp.307-320.

[KSR92] U. Kuebschull, E. Schubert, W. Rosenstiel, "Multilevel logic synthesis based on functional decision diagrams", *IEEE EDAC*, pp. 43-47, Mar. 1992.

[Krst98]A. Krsti•, K.-T. Cheng, "Delay fault testing for VLSI circuits", *Kluwer Academic Publishers*, 1998.

[Lai93] Y.-T. Lai, M. Pedram, S. B. Vrudhula, "FGILP: An integer linear program solver based on function graphs", *Proc. IEEE/ACM ICCAD*, pp. 685-689, Nov. 1993.

[Lee94] J. Lee, J.H. Patel, "Architectural level test generation for microprocessors", *IEEE Trans. CAD*, vol.13, no.10, pp.1288-1300, Oct. 1994.

[Lev98] R. Leveugle, R. Ubar, „Synthesis of DDs from clock-drivaen multi-process VHDL for test generation", *Proc. MIXDES'98*, pp. 353-358, Lodz, Poland, June 1998.

[LL92] H.-T. Liaw, C.-S. Lin, "On the OBDD-representation of general Boolean functions", *IEEE Trans. on Computers*, Vol. C-41, No. 6, pp. 61-664, June 1992.

[Luo98] Y. Luo, T. Wongsonegoro, A. Aziz: «Hybrid Techniques for Fast Functional Simulation», *DAC 1998.*

[MB88] J. C. Madre, J. P. Billon, "Proving circuit correctness using formal comparison between expected and extracted behavior*", Proc. 25th ACM/IEEE DAC*, pp. 205-210, June 1988.

[Mei74] K. C. Y. Mei, "Bridging and stuck-at faults", *IEEE Trans. Comput.*, vol. C-23, no. 7, pp. 720-727, July 1974.

[Meno65] P. R. Menon, "A simulation program for logic networks", *Bell Laboratories Internal Memorandum*, Mar.1965.

[Min93] S. Minato, “Zero-suppressed BDDs for set manipulation in combinatorial problems”, *Proc. 30th ACM/IEEE DAC*, pp. 272-277, June 1993.

[Min96] S. Minato. Binary Decision Diagrams and Applications for VLSI CAD. Kluwer Academic Publishers, 1996, 141 p.

[MMS95] P. McGeer, K. McMillan, A. Saldanha, A. Sangiovanni-Vincentelli, P. Scaglia: «Fast Discrete Function Evaluation Using Decision Diagrams», *ICCAD 1995*, pp 402-407

[Mur88] B. T. Murray, J. P. Hayes, "Hierarchical test generation using precomputed tests for modules", *Proc. Int. Test Conf.*, pp. 221-229, 1988.

[Nie91] T.M. Niermann, J.H. Patel, "HITEC: A test generation package for sequential circuits", *Proc. of EDAC*, 1991, pp.214-218.

[Pom96a] I. Pomeranz, S. M. Reddy, "On static compaction of test sequences for synchronous sequential circuits". *Proc. Design Automation Conf.*, 1996, pp. 215-220.

[Pom96b] I. Pomeranz, S. M. Reddy, "Dynamic test compaction for synchronous sequential circuits using static compaction techniques". *Proc. IEEE Fault Tolerant Computing Symp.*, 1996, pp. 53-61.

[Pom97] I. Pomeranz, S. M. Reddy, "Vector restoration based static compaction for synchronous sequential circuits". *Proc. of ICCD*, 1997, pp. 360-365.

[Rai00] J. Raik, R.Ubar. Fast Test Pattern Generation for Sequential Circuits Using Decision Diagram Representations. Journal of Electronic Testing: Theory and Applications, Kluwer Academic Publishers. Vol. 16, No. 3, pp. 213-226, June, 2000.

[Rai99a]J.Raik, R.Ubar. Sequential Circuit Test Generation Using Decision Diagram Models, *Proceedings of the DATE Conference*, pp. 736-740, Munich, Germany, March 9-12, 1999.

[Rai99b]J.Raik, R.Ubar. High-Level Path Activation Technique to Speed Up Sequential Circuit Test Generation, *Proc. of the European Test Workshop*, pp. 84-89, Konstanz, Germany, May 25-28, 1999.

[Rud94] E. M. Rudnick, J. H. Patel, G. S. Greenstein, T. M. Niermann, “Sequential Circuit Test Generation in a Genetic Algorithm framework,” Proc. Design Automation Conf., pp. 698-704, 1994.

[Rud97] E. M. Rudnick, J. H. Patel, "Putting the squeeze on test sequences".

Proc. of ITC, 1997, pp. 723-732.

[Sesh65] S. Seshu, "On an improved diagnosis program", *IEEE Trans. On Electron. Comput.*, vol. EC-14, pp. 76-79, 1965.

[SKMB90] A. Srinivasan, T. Kam, S. Malik, R. Brayton, “Algorithms for discrete function manipulation”, *Proc. IEEE/ACM ICCAD*, pp. 92-95, Nov. 1990.

[Tha80] S. M. Thatte, J. A. Abraham, “Test Generation for Microprocessors”, *IEEE Tran. Comp.*, vol. c-29, pp. 429-441, Jun. 1980.

[Tho96] K. M. Thomson, "Intel and the myths of test". *IEEE Design & Test of Computers*, Spring 1996, pp. 79-81.

[Timo83] C. Timoc, F. Scott, K. Wickman, L. Hess, "Adaptive self-test for a microprocessor", *Proc. International Test Conference*, pp. 701-703, October 1983.

[Uba76] R.Ubar, “Test Generation for Digital Circuits Using Alternative Graphs”, *Proc. of Tallinn Technical University*, Estonia, No. 409, pp. 75-81 (in Russian), 1976.

[Uba79] R.Ubar, “Alternative Graphs and Test Generation for Digital Systems”, *Proc. 2nd Conf. On Fault Tolerant Systems and Diagnostics*, pp. 177-184, Brno, Czechoslovakia, 1979.

[Uba83] R.Ubar, “Test Pattern Generation for Digital Systems on the Vector Alternative Graph model”, *13-th International Symposium on Fault Tolerant Computing*, Milano, Italy, pp. 347-351, 1983.

[Uba94] R. Ubar, “Parallel Critical Path Tracing Fault Simulation,” *Proc. of the 39. Int. Wiss. Kolloquium*, Ilmenau, Germany, Sept. 27-30, 1994. Band 1, pp. 399-404.

[Uba98] R. Ubar, “Multi-Valued Simulation of Digital Circuits with Structurally Synthesized Binary Decision Diagrams,” *OPA (Overseas Publishers Assotiation) N.V. Gordon and Breach Publishers, Multiple Valued Logic*, Vol.4, 1998, pp. 141-157.

[Uba98b] R.Ubar. Combining Functional and Structural Approaches in Test Generation for Digital Systems. *Journal of Microelectronics and Reliability*, Elsevier Science Ltd. Vol. 38:3, pp.317-329, 1998.

[UJP01]R. Ubar, A. Jutman, Z. Peng, “Timing Simulation of Digital Circuits with Binary Decision Diagrams,” *Proc. of DATE 2001 Conference*, München, Germany, March 13-16, 2001, pp. 460-466.

[UK01] R.Ubar, W.Kuzmicz, W.Pleskacz, J.Raik. Defect-Oriented Fault Simulation and Test Generation in Digital Circuits, International Symposium on Quality of Electronic Design, March 26-28, San Jose, California, USA.

[Ulri65] E. G. Ulrich, "Time-sequenced logic simulation based on circuit delay and selective tracing of active network paths", *Proc. 20th ACM Natl. Conf.*, pp. 437-448, 1965.

[UMR99] R. Ubar, A. Morawiec, J. Raik, «Cycle-based Simulation with Decision Diagrams», *Design Automation and Test in Europe (DATE) Conference 1999*, München, Germany, March 9-12, 1999, pp 454-458.

[Ward90] P. C. Ward, J. R. Armstrong, "Behavioral fault simulation in VHDL", *Proc. 27th Design Automation Conf.*, pp. 587-593, June 1990.

[Will81] T. W. Williams and N. C. Brown, "Defect Level as a Function of Fault Coverage", *IEEE Trans. on Computers*, Vol. C-30, No. 12, pp. 987- 988, 1981.