Czech Technical University in Prague
Faculty of Information Technology
Department of Digital Design

Implicit Representations in Testing and Dependability of Digital Circuits

by

Jiří Balcárek

A dissertation thesis submitted to
the Faculty of Information Technology, Czech Technical University in Prague,
in partial fulfilment of the requirements for the degree of Doctor.

Dissertation degree study programme: Informatics

Prague, August 2016

Supervisor:
doc. Ing. Jan Schmidt, Ph.D.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Co-Supervisor:
doc. Ing. Petr Fišer, Ph.D.
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract and contributions

This doctoral thesis deals with application of implicit representations of vector sets in the field of digital circuits testing and dependability evaluation. Here, an algorithm performance and output quality significantly depend on the representation of vector sets. Explicit representations of vector sets (e.g. enumeration of vectors) have been thoroughly studied in the past, but they are not suitable for large industrial circuits with huge sets of vectors. Thus, late research aims on more scalable representations of vector sets. In this field, the most promising way is to describe a vector set by its characteristic function, which can be implicitly represented by Boolean networks (or their special cases like And-Inverter Graph), Boolean cubes, SATisfiability (SAT) instances in Conjunctive Normal Form (CNF) or Binary Decision Diagrams (BDDs) and their modifications, etc. This thesis summarizes the most common implicit representations of a vector set and shows some breakthroughs or suggests interesting ways for further proceeding in different fields of digital system design, testing and verification.

The implicit representation of a vector set as a SAT instance in a CNF seems to be most promising and scalable in the field of digital circuit testing and dependability evaluation. Thus, this representation of a vector set is examined in detail. An extensive research on properties of SAT instances in a CNF confirms some previous observations and introduces new ones. The application of this implicit representation is experimentally evaluated on two pilot examples:

1. The SAT-Compress algorithm shows an application of the SAT-based approach in the field of serial compression of test vectors. Even a simple greedy algorithm is able to reach the compression ratio comparable with the state-of-the-art tools (86-90% on average). The influence of initial conditions, simple heuristics and speedup techniques on the test compression performance is discussed, as well as the suitability of the implicit representation of the vector set for similar problems.

2. The fault classification algorithm shows an application of the SAT-based approach in the in the field of dependability evaluation, where faults must be quickly and precisely classified into four groups by their observability on the circuits outputs.

This algorithm utilizes an implicit representation of vectors for proving arbitrary predicates quantified over an input vector. This method combines features from SAT-based Automatic Test Pattern Generation (ATPG) and SAT-based property checking.

Moreover, a novel and original method to convert a conceptual hardware (miter) to a Pseudo Boolean Optimization (PBO) problem instance was proposed. This technique allows to produce test patterns with a high number of don't care (unspecified) bits. Our extensive research on don't care bits in serial compression disapproved an old thesis, which claims that more don't care bits in test vectors grants a compression algorithm much more freedom in overlapping and grants a better compression.

Keywords: implicit representations, testing, test compression, dependability

Acknowledgements

First of all, I would like to thank my supervisor Dr. Jan Schmidt and my co-supervisor Dr. Petr Fišer, who always encouradged me in my research and led me to produce precise and meaningfull scientific results. Specially, I would like to thak Dr. Petr Fišer for enourmous patience in reading and corrections of my scientific papers.

This research was partially supported by the Ministry of Education, Youth, and Sport of the Czech Republic under research program MSM 6840770014, by the Czech Science Foundation as project No. 102/09/1668, and by students grants of the Czech Technical University in Prague SGS10/118/OHK3/1T/18, SGS11/089/OHK3/1T/18, SGS12/094/OHK3/1T/18, SGS13/101/OHK3/1T/18, SGS14/105/OHK3/1T/18.

Contents

List of Figures

List of Tables

Abbreviations

AIG	And-Inverter Graphs
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BDD	Binary Decision Diagram
CATAPULT	Concurrent Automatic Testing Allowing Parallelization and Using Limited Topology
CED	Concurrent Error Detection
CL	Coverage Loss
CNF	Conjunctive Normal Form
COMPAS	Compressed Test Pattern Sequencer for Scan Based Circuits
CPDCI	Coverage Preserving Don't Care Injection
CTPG	Constraint Test Pattern Generation
CUT	Circuit/Core Under Test
DAG	Directed Acyclic Graph
DFS	Depth First Search
D&P	Davis&Putnam algorithm
DC	Don't Care
DSOP	Disjoint Sum-Of-Product
EDA	Electronic Design Automation
EDT	Embedded Deterministic Test
ESOP	EXOR-Sum-Of-Product
ETC	Embedded Tester Core
EVBDD	Edge-Value Binary Decision Diagram
FL	Fault List
FS	Fault Security
ILP	Integer Linear Programming
LUT	Look Up Table
MDS	Modified Duplex System
NFS	Not Fail Safe

NST	Not Self-Testing
OBDD	Ordered Binary Decision Diagram
PBO	Pseudo Boolean Optimization
PDF	Path Delay Fault
PI	Primary Input
PO	Primary Output
PODEM	Path Oriented Decision Making
RESPIN	REusing Scan chains for test Pattern decompression
ROBDD	Reduced Ordered Binary Decision Diagram
SAT	Satisfiability
SEU	Single Event Upset
SoC	System-on-Chip
SOCRATES	Structure-Oriented Cost-Reducing Automatic TESt pattern generation system
SPIRIT	Satisfiability Problem Implementation for Redundancy Identification and Test generation
SSBDD	Structurally Synthetized Binnary Decision Diagram
SSMIBDD	Structurally Synthetized Multiple Input BDD
ST	Self-Testing Property
TAM	Test Access Mechanism
TDF	Transition Delay Faults
TPG	Test Patterns Generation
TP	Test Pattern
TSC	Totally Self-Checking
TSP	Traveling Salesman Problem
UNSAT	UNSATisfiable

Chapter 1

Introduction

Testing of digital circuits and their dependability are of a great concern in circuit design. Testing significantly influences the time to market and yield, while the dependability is crucial in safety-critical systems.

The complexity of digital circuits and systems increases and the ability of computing devices to solve problems like test generation and dependability evaluation in a reasonable time significantly depends on a chosen algorithm and search space representation. Universal solutions become inefficient, thus solutions customized for a specific task are preferred. Researchers all around the world are forced to optimize algorithms for circuit synthesis, test generation, dependability evaluation and other tools globally used in EDA (Electronic Design Automation).

In this context, the search space representation is crucial for further processing and it directly influences the time and memory consumption. Algorithms for test generation for digital circuits, as well as algorithms for computing of dependability parameters must usually deal with huge sets of choices to be processed. The sets express either the possibilities of choice (at least one vector from the set is needed), or summary (the whole set of vectors is needed). Such sets can be expressed explicitly, as an enumeration of possible choices, or implicitly, by describing the required characteristic properties of the set.

In EDA tools processing logic circuits, an explicit representation (enumeration) of the vectors set is usually infeasible (memory, CPU computation time, etc.), because the number of vectors often grows exponentially with the circuit size (the number of gates) or the number of primary inputs (PIs). Hence, using implicit representations is necessary.

Implicit representations of a vector set allow EDA tools to solve huge problems. However, generation of characteristic properties of the set and operations over them can be also very time or memory consuming in general. Hence, the ability to solve a problem depends on properties of each instance, the chosen representation, and the algorithm applied. An appropriate choice can prevent a time or memory explosion, e.g., as in satisfiability (SAT) based Automatic Test Pattern Generation (ATPG) tools. Here, the set of test patterns is represented as a SAT problem instance in a Conjunctive Normal Form (CNF) [1],[2]. Another example is a precise dependability evaluation. The algorithm must process all possible configurations of the circuit primary inputs and their responses on primary outputs (POs) [3],[4]. This thesis also discusses the application of SAT ATPG principles in the field of test patterns compression and dependability evaluation.

1.1 Motivation

The digital circuit design, testing and dependability evaluation are areas which have been thoroughly studied for many decades and various techniques and algorithms were presented. Most of these techniques are designed to directly process a table of vectors [5],[6],[7],[8],[9], state set [10], or perform operations by traversing the circuit [11],[12],[13],[14],[15],[16]. These techniques were able to reach remarkable results, but lately, with the growing circuit size, their performance can significantly decrease. This degradation is often caused by the amount of information to be processed, which typically grows exponentially with the circuit size. Recent techniques and algorithms try to overcome this performance degradation by application of new structures or more sophisticated algorithms.

In this context, the application of implicit representations of the information caused a great breakthrough for many painful problems like model checking [10], circuit verification [17],[18], testing of hard-to-test faults [2],[19], path delay faults test generation [20],[21], diagnosis of the circuit considering multiple faults [22],[23],[24], and many others (chapter 3, the State-of-the-Art for details). In most cases, the processed information is the set of vectors obtained by some transformation from a digital circuit. This set of vectors is typically encoded as Binary Decision Diagrams (BDDs) [25],[26], their modifications [27],[28],[29], or a SATisfiability (SAT) problem instance in a Conjunctive Normal Form (CNF).

Previous achievements reached by application of implicit representations of vector sets encouraged us to make further research on the most popular implicit representations and to show their possible applications. Digital circuit's dependability and testability is of a great concern of every engineer, and thus the case studies presented in this Thesis are focused on this area. The application of implicit representations of vector sets in test vectors generation proved to be very efficient mainly for hard-to-test-faults. Thus, we explore further utilization in test vectors compression in this Thesis too. Moreover, achievements in the field of model checking and circuit verification suggest that implicit representations could be beneficial for dependability computation tasks where some predicate must be evaluated for all vectors on circuits' primary inputs (PIs).

1.2 Problem Statement

This Thesis aims at implicit representations of vector sets and their application in serial compression [5],[6],[7],[30] and dependability evaluation [3],[4],[31]. The SAT or BDD based ATPGs proved to be highly efficient in tasks where the set of test vectors is small (hard-to-test faults) [2],[19] or the whole set or a suitable subset of test vectors is needed (constrained vectors) [19],[32]. The main aim of this Thesis is to show interesting properties of widely used implicit representations like SAT in CNF, BDD etc. and to try to show their strengths and weaknesses on two case studies.

This thesis focuses on:

- Research on properties of implicitly represented vectors sets. These properties may indicate how the representations can behave during their processing. These observations and properties can be utilized for miscellaneous design and optimization algorithms.

- Case study on serial compression of the vectors set. Serial compression is typically based on a greedy algorithm which tries to compress (find maximum overlapping) test vectors pre-generated by an ATPG [5],[6],[7],[30]. The pre-generated set of test vectors significantly influences the compression rate. It is impossible to enumerate all possible test vectors for all faults and choose the best subset to maximize their overlap and thus maximize their compression rate. This seems to be a suitable task for application of implicit representations. The main idea is that the set of test vectors for each fault can be represented implicitly and the best vector for overlap (maximizing the overlapping) can be chosen on-the-fly by applying constraints. This research is a direct extension of SAT-based ATPG techniques. The theoretical outcomes can be partially used in tasks, where some constrained subset of test patterns is generated, like path delay faults test generation [33],[20],[21], multiple fault diagnosis [22],[23],[24], power aware test generation [32], and many others.

- Case study on application of implicit representations for dependability evaluation. Dependability seems to be another suitable area to demonstrate the strengths and weaknesses of implicit representations. Its precise evaluation frequently depends on evaluation of some predicate over the whole set of input vectors. Application of the whole vectors set on primary inputs is infeasible for larger circuits, because the number of test vectors grows exponentially with the number of the circuit's primary inputs (PIs) [3],[4]. Thus, our research is focused on evaluation of general predicates over the implicitly encoded set of input vectors. This research generalizes the principles used in SAT ATPGs and demonstrates its advantages and disadvantages in fault classification. The main aim is to speed up the precise evaluation of dependability parameters. The theoretical outcomes can be partially used in tasks where a general predicate must be evaluated over a set of vectors, like fault classification and dependability evaluation [3],[4],[31], multiple fault diagnosis [22],[23],[24] and many others.

Simply summarized, this thesis discusses the properties of implicit representations and their application in pilot examples (serial compression/fault classification) and forms basic rules for their application in similar tasks. The pertinence of test vector representations and the way of processing for these problems is discussed as well.

1.3 Contributions of the Thesis

The main aim of this thesis is to discuss the application of implicit representations of vector sets in serial compression and dependability evaluation. This thesis sumarizes application of implicit representations of the vectors set in different fields of digital system design, testing and verification (chapter 3). Moreover, two pilot examples are introduced:

- SAT-Compress [A.10],[A.9],[A.17],[A.5] has been introduced as a pilot example for serial compression. A simple and easy to implement SAT-based compression algorithm which is able to reach the compression ratio comparable with state-of-the-art algorithms (86-90% on average).
- A fault classification [A.6],[A.8],[A.12] tool was introduced as a pilot example for a fast and precize fault classification. This tool utilizes an implicit representation of vectors for proving arbitrary predicates quantified over an input vector of a combinational circuit. This method combines features from SAT ATPG and SAT based property checking.

Furthermore, an extensive research on SAT instances produced by Tseitin transformation from combinational circuits has been performed [A.11]. These measurements show that:

- SAT instances are similar to 2+p-SAT problems, where p = 2.37, which makes them easy-to-solve (confirms [34]).
- SAT instances can be dramatically reduced, while the represented test vectors set is preserved (60% variables and 65% terms can be removed on average).
- Satisfiability of these instances is 99% on average, while their random clones have only 10% of satisfiable instances on average.
- SAT instances are highly constrained. On average 57% of variables are fixed to a constant value for every SAT solution (backbone).

Moreover, research on serial compression disapproved an old thesis [6],[35],[36], which claims that more don't care bits in test vectors grants a compression algorithm much more freedom in overlapping and grants a better compression [6],[35].

Research on pilot examples proved, that implicit representations of the vector set can be very beneficial for problems where the vectors or their subset are required (fault classification), but to some extent it fails for problems where the output is a sequence of vectors (serial compression).

Techniques to increase the efficiency of SAT-based serial compression have been proposed and discussed [A.5],[A.1],[A.2],[A.7].

A novel and original method to convert a conceptual hardware (miter) to a Pseudo Boolean Optimization (PBO) instance was proposed [A.1],[A.2].

Numerous experiments were performed to obtain precise and reliable data.

1.4 Structure of the Thesis

This thesis is organized into eight chapters as follows:

1. *Introduction*: Discusses the current situation in the field of digital system design, test and verification. Summarizes the motivation for our research and forms the problem statement. Finally, a brief overview of contributions is introduced.

2. *Theoretical Background*: Defines and clarifies theoretical terms and definitions generally used in this thesis.

3. *State-Of-The-Art*: Here, the research on implicit representations of vectors sets is summarized. Representative techniques from different fields of digital system design, test and verification are briefly described. This chapter shows the importance of implicit representations of vectors set in this field and breakthroughs given by their application, or interesting techniques of vectors processing.

4. *Properties of Implicitly Represented Vectors Set*: Summarizes and discusses the research on properties of implicit representation of the vectors set as a SAT instance in CNF.

5. *Serial Compression*: Summarizes and discusses the research on serial compression. A dedicated sub-chapter *"state-of-the-art"* summarizes current compression techniques and addresses their strengths and weaknesses. The SAT-Compress algorithm is introduced as a case study. New technique for conversion of conceptual hardware to a PBO instance is introduced. Experimental results show properties of the serial compression problem and demonstrate the behavior of the SAT-based algorithm. Discusses effects which influence the compression ratio and scalability.

6. *Dependability and Fault Classification*: Summarizes and discusses application of SAT-based techniques to fault classification. First, the target architecture is described. The fault classification and techniques of dependability evaluation are introduced. A dedicated sub-chapter *"state-of-the-art"* summarizes current techniques and addresses their strengths and weaknesses. A method for proving arbitrary predicates quantified over an input vector of a combinational circuit is presented. Experimental results are discussed to show strengths and weaknesses of the SAT-based approach.

7. *Discussion*: Discusses case studies introduced in chapter 5 and chapter 6. Targets their suitability for application of implicit representations of the vectors set and forms general recommendations for their manipulation.

8. *Conclusions*: Recapitulates contributions of this thesis, summarizes the results and discusses a possible future work.

Chapter 2

Theoretical Background

This chapter defines terms and principles generally used in this thesis. Definitions specific for case studies are described in dedicated chapters as *preliminaries*.

2.1 Preliminaries

First let's define some basic terms used in this thesis.

A **Boolean function** $f(x) = f(x_1, \ldots, x_n)$, where n denotes the number of input variables, describes a mapping $B^n \to B$ between input and output Boolean domain $B = \{0, 1\}$, thus $f : \{0, 1\}^n \to \{0, 1\}$. In fact, a Boolean function determinates the output value $\{\mathit{0},\ \mathit{1}\}$ for a vector of Boolean variables (or constants) given as input.

A **characteristic function** χ_f of a Boolean function *f(x)* is a special case of Boolean function, which denotes if the input vector of variables and the output value of *f(x)* respect the mapping given by *f(x)* (consistency check), thus $\chi_f(x_1, \ldots, x_n, f(x)) = 1 \Leftrightarrow f(x) = f(x_1, \ldots, x_n)$.

From our point of view, a Boolean function represents a set of Boolean vectors where each vector consists of input variables and its characteristic function denotes if a given Boolean vector belongs to this set.

Each Boolean function can be described by a Boolean network (see subsection 2.1.1), a Boolean formula (general or in a conjuctive/disjunctive normal form), truth table (enumeration of vectors), Binary decision diagram, or many others (see section 2.3). Each representation of such vector set (Boolean function) has specific properties like the size and hardness of processing, which determinate their applicability in practical tasks. Our research aims on these properties and studies the applicability of representations of the vector set in serial compression and dependability evaluation.

2.1.1 Boolean Network

The most common input of EDA tools is a netlist of gates which represents a combinational circuit to be processed. This netlist of gates can be modeled as a Boolean network. A

Boolean network is a Directed Acyclic Graph (DAG) with vertrices (nodes) corresponding to logic gates and edges corresponding to wires (signals) interconnecting particular gates. An example is shown in Fig. 2.1. Inputs x_1, x_2, x_3, x_4 of the Boolean network are called Primary Inputs (PIs) and outputs z_1, z_2 are called Primary Outputs (POs). A node f_x is called a fanin node of a node f_y, if there is a directed edge from f_x to f_y and a fanout node if there is a directed edge from f_y to f_x. A node f_x is a *transitive fanin node* of a node f_y if there is a directed path from f_x to f_y and a *transitive fanout node* if there is a directed path from f_y to f_x. Each node in a Boolean network has a logic function associated (e.g. basic gate function AND, OR, NOT, XOR), which corresponds to the represented logic gate.

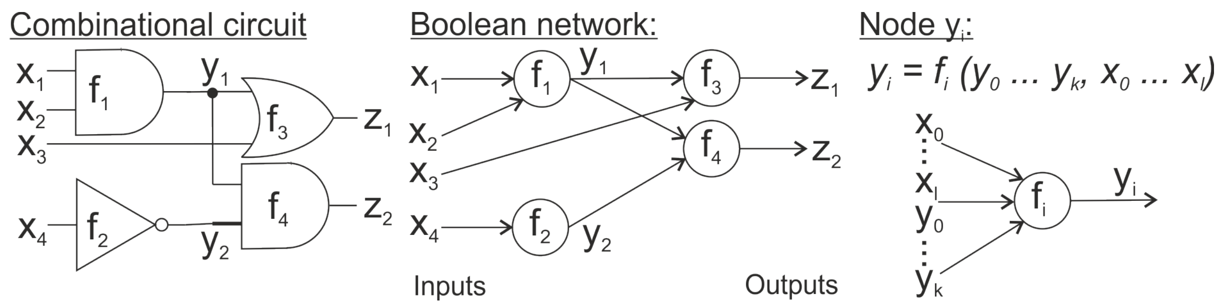

Figure 2.1: Boolean network examples.

2.1.2 CNF Satisfiability Problem

Let us have a Boolean formula given in a Conjunctive Normal Form (CNF), i.e., written as a conjunction of clauses, where each clause consists of a disjunction of literals. A literal is a logic variable or its complement.

Definition 1. Deciding whether a formula in CNF is satisfiable is called a satisfiability (SAT) problem. If the formula consists only of clauses having exactly k literals, we call the problem *k-SAT*.

Example 1. Let us have a formula in CNF: $\varphi = (a \vee b \vee \neg c) \wedge (\neg a) \wedge (a \vee c)$. The formula is called satisfiable, when there exists an assignment of variables *a, b, c*, so that the formula is equal to *1*. We can see that φ is equal to one for a combination *(a, b, c) = (0, 0, 1)*, thus it is satisfiable.

It is well known that the *k-SAT* problem is NP-complete for $K \geq 3$ [37]. The *2-SAT* problem is solvable in a polynomial time [38].

Definition 2. Deciding on satisfiability of CNF formulae having a mixture of 2- and 3-literal clauses is called a *2+p-SAT problem* [39]. Here p stands for a percentage of 3-literal clauses. This model interpolates between *2-SAT* (for p = *0*) and *3-SAT* (for p = *1*) problems. For p >*0* the *2+p-SAT problem* is NP-complete as well.

2.2 Implicit vs. Explicit Representations of Vector Sets

This chapter clarifies the definition of implicit and explicit representations of the vector set and shows typical representatives of both.

In context of cognitive science, the information representation can be denoted as implicit or explicit [40]. The information is *explicitly* represented (encoded), if it is expressed literally and unambiguously and can be extracted quickly by a constant-time algorithm, while information is *implicitly* encoded if it is not accessed directly and additional processing is needed to recover it. It means that cognitive science considers only *accessibility* of encoded information.

In context of digital circuits design, verification and testing, the *implicit* and *explicit* representations of the vector set are expanded as follows:

- The vector set is represented explicitly, if particular vectors can be extracted directly without additional processing and the size of the representation grows proportionally with the number of vectors in the set. This is, e.g., an enumeration of vectors.
- The vector set is represented implicitly if it is described by some characteristic properties, thus a vector cannot be extracted without additional effort and the size of the representation does not grow proportionally with the number of vectors in the set. The set can be represented as, e.g., a satisfiability problem (SAT) instance in a Conjunctive Normal Form (CNF) or a Reduced Ordered Binary Decision Diagram (ROBDD).

In fact, the expanded definition of implicit and explicit representation of a vector set introduces dependency on vector representation (encoding) size. More precisely, the relation between the size of the vector set and its encoded representation is considered, as well as the vector accessibility.

It can be concluded that the expanded definition of implicit representation of the vector set introduces a reduction of memory consumption, but it increases the complexity of processing. The main aim of researchers is to find a balanced ratio between memory and time consumption, while maintaining performance.

The most common representation of a vector set in the field of digital system design, testing and verification is an enumeration of vectors. Such enumeration of vectors represents the vector set explicitly and it is not suitable for large sets, because memory requirements or simple iteration of vectors are infeasible. Thus, current algorithms try to describe a vector set by a characteristic function. Furthermore, the characteristic function can be implicitly represented by Boolean networks (or their special cases like And-Inverter Graphs (AIG) [41]), Boolean cubes, SAT instances in CNF or Binary Decision Diagrams (BDDs) [27], [26] and their modifications. Comprehensive survey of particular implicit representations of the vector set can be found in the next chapter.

2.3 Implicit Representations of Vector Set

The previous chapter defines implicit and explicit representations of vectors sets. The represented set of vectors can be implicitly described by a characteristic function. A very suitable and straightforward way of characteristic function construction in the field of digital design, testing and verification is by introduction of *"conceptual hardware"*. Conceptual hardware (which is typically a logic circuit) is often created from the processed digital circuit, which is extended by additional logic representing assertions about the circuit [1], [22], [23], [24], [31], [42]. Conceptual hardware is not intended for synthesis, but describes the desired characteristic function of the vector set. Conceptual hardware itself is in fact an implicit representation of the vector set, but it is often transformed to more suitable implicit representations of the vector set, e.g., a SAT instance in CNF or a Binary Decision Diagram (BDD). These are more suitable for formal verification methods, which are very efficient for proving or disapproving of given assertions.

This chapter describes and compares the most common implicit representations of vector sets (characteristic function, conceptual hardware), like a set of Boolean cubes, a SAT problem instance in CNF, And-Inverter Graph (AIG) [41], Binary Decision Diagram (BDD) [27], [26] and its modifications like Ordered Binary Decision Diagram (OBDD) [27] or Reduced Ordered Binary Decision Diagram (ROBDD) [27].

Fig. 2.2.a shows a Boolean function (vector set) represented by a truth table. In this case, the truth table explicitly represents a vector set. Each particular vector of the vector set is represented directly by a row of this truth table. Each column of the truth table represents a bit in the vector, which is labeled by a name given in the header row. A Boolean function (or a set of vectors) can be described by its characteristic function denoting whether a given assignment of inputs, outputs and internal signals is consistent or inconsistent with a particular gate function. The characteristic function of the vector set is shown in Fig. 2.2.b. This characteristic function $\chi_f(x_1, x_2, x_3, f) = 1$ if the vector v = {*x1, x2, x3, f*} belongs to the set of vectors enumerated in table shown in Fig. 2.2.a.

Representation of a vector set by Boolean cubes (see. Fig. 2.3.a) can be considered as an implicit representation of the vector set if a particular Boolean cube represents (encodes) more than one vector. These Boolean cubes can be extracted from the truth table by recursive application of Boolean adjacency rules. Two vectors are adjacent if they differ in one bit only. For example, vectors *[110]* and *[111]* can be encoded to vector *[11X]* where *'X'* represents don't care value. The don't care value can be replaced by logic value *'0'* or *'1'*. Thus, introducing don't care values produces a Boolean cube, which implicitly represents a set of vectors.

Fig. 2.3.b shows a representation of the vector set by a Boolean formula. In practical tasks, a Boolean formula is often described in a Conjunctive Normal Form (CNF), see Fig. 2.3.c. Each variable assignment, which satisfies conditions described by a Boolean formula in CNF, represents one vector from a vector set. A set of all vectors can be explicitly described as enumeration of vectors, which satisfy the Boolean formula in CNF. The set of clauses defines constraints of the variables assignments. Thus, the CNF implicitly represents a vector set satisfying constraints. The instance of CNF can be easily obtained

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

a)

x_1	x_2	x_3	f	χ_f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1

b)

Figure 2.2: Truth tables a) Boolean function (vector set) b) Characteristic function of Boolean function.

x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	x	0
1	1	x	1

a)

$$\begin{aligned}
&(\neg x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge f) \vee \\
&(\neg x_1 \wedge \neg x_2 \wedge x_3 \wedge \neg f) \vee \\
&(\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge \neg f) \vee \\
&(\neg x_1 \wedge x_2 \wedge x_3 \wedge f) \vee \\
&(x_1 \wedge \neg x_2 \wedge \neg f) \vee \\
&(x_1 \wedge x_2 \wedge f)
\end{aligned}$$

b)

$$\begin{aligned}
&(x_1 \vee x_2 \vee x_3 \vee f) \wedge \\
&(x_1 \vee x_2 \vee \neg x_3 \vee \neg f) \wedge \\
&(x_1 \vee \neg x_2 \vee x_3 \vee \neg f) \wedge \\
&(x_1 \vee \neg x_2 \vee \neg x_3 \vee f) \wedge \\
&(\neg x_1 \vee x_2 \vee x_3 \vee \neg f) \wedge \\
&(\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg f) \wedge \\
&(\neg x_1 \vee \neg x_2 \vee x_3 \vee f) \wedge \\
&(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee f)
\end{aligned}$$

c)

Figure 2.3: Implicit representation of the vector set: a) Boolean cubes b) characteristic function c) characteristic function in CNF.

from a Boolean network [1], [43], [44], [45], [46] and its size grows linearly with the circuit size.

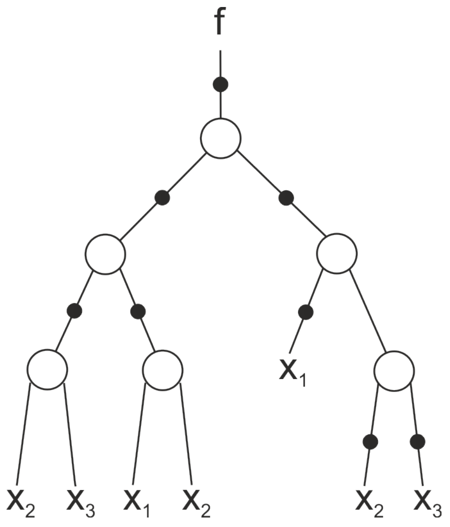

Figure 2.4: Boolean network represented as And-Inverter Graph (AIG).

And-Inverter Graph (AIG) [41] is a special case of a Boolean network. It is represented as a Directed Acyclic Graph (DAG), see Fig. 2.4. It consists of one primary output (for the case of characteristic function), primary inputs, two-input AND gates and inverters. Each AND gate is represented by a node with two inputs and the inversion is indicated by a marker on the edge. Time consumption of the conversion from any Boolean network is proportional to the size of the circuit. Thus, it is fast and scalable. This representation of a characteristic function is very suitable for further conversion to CNF. AIG consists of AND gates and inverters. Thus, the CNF instance produced from AIG is more homogenous which is beneficial for further processing (e.g. solving by SAT solver, fault simulation, etc.) [47], [48].

Binary Decision Diagrams (BDDs) [27], [26] are also a very popular representation of the vector set. Here, the vector set (characteristic function) is represented as a rooted Directed Acyclic Graph (DAG) with a root node $G = (V, E)$, where V is a set of nodes, while E is edge set. It consists of decision nodes (internal) and terminal nodes (*0-terminal* and *1-terminal*). Nodes are labeled by variable names and are connected by directed edges which are labeled with a logical value representing the assignment of logical value (*0 = low*, *1 = high*) to the node (variable). Terminal nodes have no child nodes and they are labeled with a value *0* or *1*. Each internal node in BDD has exactly two successors (e.g. node u has *low(u)* and *high(u)*) and each path from the root to any terminal node consists of all variables. Thus, BDD is in fact a binary decision tree, see Fig. 2.5.a. Such a binary decision tree is not very efficient for representation of larger characteristic functions, because the number of nodes always grows exponentially with the number of variables. Thus, additional rules were introduced to increase the efficiency of BDDs. First, the order of variables on

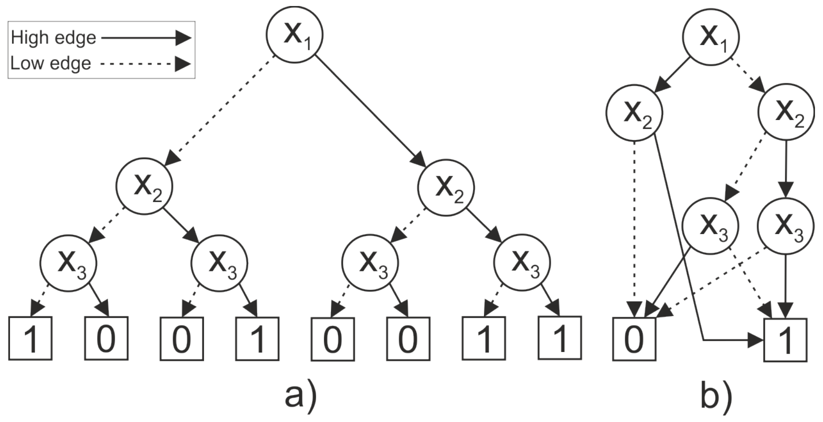

Figure 2.5: Implicit representation of the vector set a) BDD, b) ROBDD.

the path from the root to any terminal node is fixed. Such BDD is called Ordered Binary Decision Diagram (OBDD) [27]. Furthermore, Fig. 2.5.a illustrates that in BDD there is a great number of redundancies. Such redundancies can be eliminated by reduction rules:

1. Eliminate duplicate terminal nodes. Terminal nodes are compacted, thus only two terminals remain.

2. Eliminate duplicate non-terminals. Merge two nodes u, v, in cases where *low(u) = low(v)* and *high(u) = high(v)*.

3. Eliminate redundant nodes, where *low(u) = high(v)*.

OBDD which is reduced by application of the reduction rules is called Reduced Ordered Binary Decision Diagram (ROBDD) [27] (see Fig. 2.5.b). ROBDDs are much more efficient for representation of vector sets than BDDs and are widely used mainly in digital system design and synthesis [49], [50], [51], [52], [53], [54], [55]. Reduction rules can reduce the size of ROBDD significantly, but the number of nodes dramatically depends on the variable order [27], [26]. However, searching for a good variable order is a NP-complete problem [56].

2.4 Circuit-to-CNF Transformation

Transformation of a combinational circuit to a Boolean formula in a Conjunctive Normal Form (CNF) is a foundation stone of many approaches in digital circuit design and verification. The circuit itself implicitly represents a characteristic function of a vector set, but it is not a very suitable representation of vector sets for formal verification methods.

Fig. 2.6 shows a simple circuit, which will be used for demonstration of the circuit-to-CNF transformation.

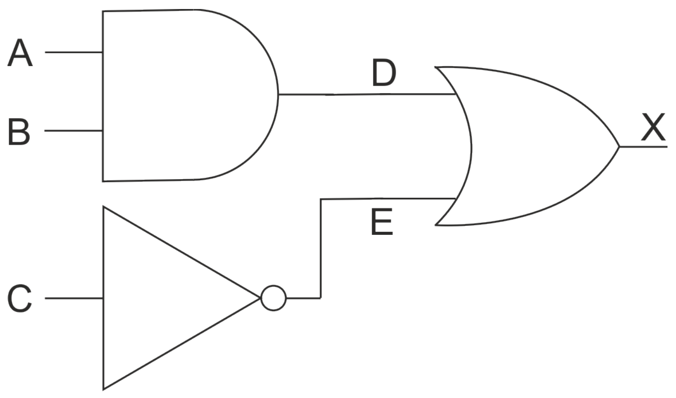

Figure 2.6: Simple combinational circuit.

There are many techniques for circuit-to-CNF transformation [1], [43], [44], [45], [46], which try to optimize the transformation process, but the most common one is the Tseitin transformation [1], [45] which is also used in this thesis. Now, its basic principles will be described.

The Tseitin transformation [1], [45] composes the circuits' CNF from characteristic functions of circuits' gates. The characteristic functions are derived from logic functions of the gates. For example, let us consider the AND gate $D = A \wedge B$. For any two functions P and Q, $P = Q$ is equivalent to $(P \Rightarrow Q) \wedge (Q \Rightarrow P)$. In this way the AND gate characteristic function is constructed as $(D \Rightarrow (A \wedge B)) \wedge ((A \wedge B) \Rightarrow D)$. Next this expression is transformed to CNF using Boolean algebra rules, obtaining $(A \vee \neg D) \wedge (B \vee \neg D) \wedge (\neg A \vee \neg B \vee D)$. The CNF for the whole circuit is constructed by conjunction of characteristic functions (in CNF) of each gate, see Fig. 2.7.

$$\begin{aligned}
&(\neg A \vee \neg B \vee D) \wedge (A \vee \neg D) \wedge (B \vee \neg D) \wedge \\
&(C \vee E) \wedge (\neg C \vee \neg E) \wedge \\
&(D \vee E \vee \neg X) \wedge (\neg D \vee X) \wedge (\neg E \vee X)
\end{aligned}$$

Figure 2.7: Circuit characteristic function in CNF.

The main advantage of the Tseitin transformation is its easy application on a Boolean network consisting of basic gates (AND, OR, NAND, NOR, NOT, BUFF). The characteristic function in CNF for each basic gate can be generically added to the circuit's CNF in linear time. The equations for generic generation of characteristic functions in CNF are shown in the last column of Table T1. Here, a Boolean variable x_i denotes a gate input and a Boolean variable y the gate output.

Table 2.1: Basic gates and their characteristic functions

Gate type	Gate function	Characteristic function in CNF
AND	$y = AND(x_1, ..., x_j)$	$\prod_{i=1}^{j}(x_i \vee \neg y) \wedge (\sum_{i=1}^{j}\neg x_i \vee y)$
OR	$y = OR(x_1, ..., x_j)$	$\prod_{i=1}^{j}(\neg x_i \vee y) \wedge (\sum_{i=1}^{j} x_i \vee \neg y)$
NAND	$y = NAND(x_1, ..., x_j)$	$\prod_{i=1}^{j}(x_i \vee y) \wedge (\sum_{i=1}^{j}\neg x_i \vee \neg y)$
NOR	$y = NOR(x_1, ..., x_j)$	$\prod_{i=1}^{j}(\neg x_i \vee \neg y) \wedge (\sum_{i=1}^{j} x_i \vee y)$
NOT	$y = NOT(x_1)$	$(x_1 \vee y) \wedge (\neg x_1 \vee \neg y)$
BUFF	$y = BUFF(x_1)$	$(\neg x_1 \vee y) \wedge (x_1 \vee \neg y)$

2.5 Automatic Test Patterns Generation (ATPG)

Despite of the fact, that the yield in digital circuit production increases, the number of defects is still considerable. A defect in implemented hardware causes a fault in the function of the digital circuit, which results as an error on primary outputs of the digital circuit.

The ATPGs are tools for generation of patterns detecting a fault (test patterns/test). The most common input of the ATPG is a digital circuit described by a Boolean network and the fault model. The output of the ATPG is a set of test patterns detecting faults which can occur in the digital circuit.

The fault model describes the effect caused by a defect in hardware implementation of the digital circuit. Such abstraction reduces the dependency on technology and the complexity of simulation (many defects can be modeled by the same fault). There are many fault models, e.g., single/multiple stuck-at fault model, bridging fault model, path/transition/gate delay fault models etc.

We can generate deterministic test patterns for all detectable faults. The core of most common deterministic ATPG procedures consists of three steps which are processed for each fault:

1. Fault activation: injects a fault to the Boolean network

2. Path sensitization: sensitize a path from the faulty signal to a primary output

3. Line justification: justify unassigned nets to fulfill the conditions given by assignments performed in previous steps.

2.6 SAT-Based Test Patterns Generation (TPG)

In SAT-based Test Patterns Generation, the TPG problem is reduced to the SAT problem. The characteristic function of the test vectors set is described by a SAT problem instance in CNF and is constructed by introduction of *conceptual hardware*. The conceptual hardware of fault-free and faulty circuits is combined and transformed to CNF [1], [2], to obtain a SAT problem instance. Satisfiable assignments of variables of this CNF are test vectors detecting the respective fault.

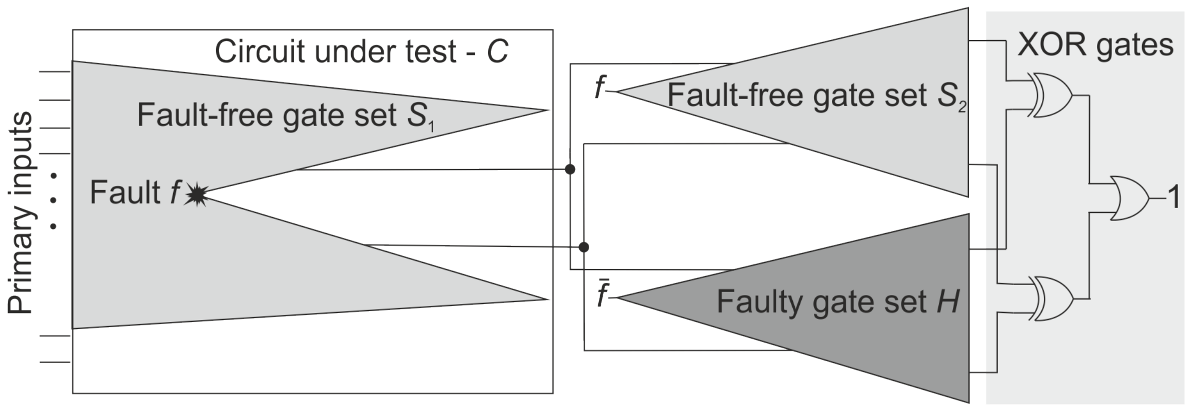

Figure 2.8: SAT instance generation for an ATPG.

A Boolean variable is assigned to each signal in the circuit. Each gate in the subcircuit S_1, which corresponds to the output cone of the fault to primary outputs (POs) of the circuit is copied and forms the set of gates H (faulty part of the circuit). All gates in the cone from the observable POs to the primary inputs (PIs) are included in the set $S \subseteq C(S = S_1 \cup S_2)$, see Fig. 2.8.

For each gate, the CNF Φ_g is derived from its characteristic function. The CNF Φ_c representing the fault-free part of the circuit is constructed as a conjunction of all CNFs of gates $gs_1, \dots, gs_n \in S$:

$$\Phi_c = \bigwedge \Phi_{gsi} \quad 1 \leq i \leq |S| \tag{2.1}$$

To generate a test for a fault F, the characteristic function Φ_f of the faulty circuit is generated as a conjunction of all CNFs of gates $gh_1, \dots, gh_n \in H$:

$$\Phi_f = \bigwedge \Phi_{ghi} \quad 1 \leq i \leq |H| \tag{2.2}$$

Outputs of the fault-free and faulty circuit are coupled by XOR gates whose outputs are further evaluated by an OR gate. The function Φ_{XOR} is generated as a conjunction of characteristic functions of these gates. The SAT instance for a fault F is obtained as a conjunction of Φ_c, Φ_f and Φ_{XOR}:

$$\Phi_{test\_F} = \Phi_c \wedge \Phi_f \wedge \Phi_{XOR} \tag{2.3}$$

Finally, 1-literal clauses are added to inject the faulty value in Φ_f, to set its image in Φ_c as a complement of the faulty value, and to set the output of the OR gate in Φ_{XOR} to *1* (the formula must result in *1* to detect the fault).

The Φ_{test_F} solutions represent the whole set of test patterns detecting the fault F. If Φ_{test_F} is unsatisfiable, the fault F is undetectable. More information can be found in [1], [2].

A conventional SAT-based ATPG algorithm [1], [2] can be described in four steps:

1. Generate a fault list for a given circuit.
2. Pick one fault from the fault list and generate its CNF.
3. Solve the SAT problem for this CNF. The solution represents a test pattern detecting the respective fault. If the CNF is unsatisfiable, the fault is removed from the fault list and marked as undetectable.
4. Simulate the test pattern obtained in step 3 and remove all detected faults from the fault list (fault dropping).
5. Repeat steps 2-4 until the fault list is empty.

For details on the SAT-based ATPGs, see [1], [2], [57], [58], [59], [60], [61], [47], [62].

Chapter 3

State-Of-The-Art

Applications of implicit representations have been thoroughly studied in past decades and caused many breakthroughs (or suggested interesting ways for further proceeding) in different fields of digital system design, testing and verification. This section briefly summarizes some problems and solutions, where implicit representations of the vector set are utilized.

3.1 Verification and Model Checking

Significant breakthrough was made in the field of model checking for finite systems [10], [48]. Previous approaches are based on decision heuristics which examine the state graph describing the system behavior. The search space (state graph) is represented by a list or a table whose size grows proportionally with the number of states. The number of states in a state graph can grow exponentially with the number of components in the system. Such a large search space cannot be explicitly stored in memory or processed in reasonable time. Thus, it is not possible to check models for large industrial systems by this approach [48].

In [10] a general method which represents the state space *symbolically* (implicitly) instead of explicitly is presented. The Mu-Calculus [63] is used as primary specification language and BDDs are used to represent relations and transitions. Even if the size of the BDD can grow exponentially, empirical results show that the BDD size grows linearly, while the number of states grows exponentially in most cases. This technique utilizing an implicit representation of state space can handle models with more than 10^{20} states. In contrast, techniques which process explicitly represented state space can handle systems with at most 10^3 to 10^6 states. The only weaknes of this approach are systems (combinational circuits like multipliers) which do not have efficient representations in BDD [27], [26]. In such cases, the BDD size explodes and the algorithm is unable to find a solution.

Further performance improvement can be achieved by a *hybrid* approach, which combines strengths of several implicit representations. Such a technique is described in [64] and aims on equivalence checking, but it can also be used for logic synthesis, timing analysis or formal property checking. This approach is unique, because it combines four different techniques: Boolean reasoning based on BDDs, structural transformations, SAT procedure, and random simulation, which are mutually orthogonal [65], and thus their strengths are combined. These techniques share the And-Inverter graph (AIG) as a Boolean network representation. First, the structural hashing over AIG is performed to solve simple problems. Then, random simulation can be efficiently used for problems with dense solution space. If structural hashing and random simulation fails to solve the problem, the BDD and SAT are used complementary. The BDD incrementally sweeps the AIG, which is simplified and thus it reduces the search space for the SAT procedure. All techniques have set threshold limits (BDD size, number of backtracks, time), which cause an abort in the current procedure and switching to another (more suitable) one. This solution seems to be very robust even for industrial circuits.

Another hybrid technique, which tries to overcome orthogonality of SAT and BDD - based approaches is introduced in [66]. The SAT-based approach is considered to be very efficient and fast, but produces only one solution, while the BDD-based approach computes all solutions in parallel, but requires a large amount of memory. Multiple solutions are sometimes desirable [67], [68], thus a hybrid approach can be beneficial. A hybrid structure based on a combination of BDD and SAT-based approach is presented. BDDs are extended by expansion nodes which have two successors representing Boolean functions, and are labeled by a Boolean operation to be performed. The algorithm starts with one expansion node which is recursively expanded based on a decision heuristic. The manner of expansion determines the appearance of the hybrid structure. For a strict Depth First Search (DFS), the algorithm performs operations similar to SAT procedure, while for symbolically carried operations it builds a BDD. The efficiency of the proposed algorithm evaluated for n-Queens and EXOR-Sum-Of-Product (ESOP) problem is encouraging, but much more experiments should be performed to prove robustness of this approach in *hard instances*.

The performance improvement of SAT-based techniques can also be achived by injection of conflict clauses [69], which can guide the decision heuristic and significantly prune the search space. In [69], the BDD is used to implicitly represent conflict clauses (see section 3.2 for more details).

3.2 Automatic Test Patterns Generation for Stuck-At Faults

Another interesting application is test vector generation for the single stuck-at fault model. This fault model supposes the existence of one fault in the combinational circuit at most. The test patterns for a fault are generated by an ATPG, see section 2.5 and section 2.6.

Previous approaches like single path sensitization [14], the D-algorithm [12], PODEM (Path Oriented Decision Making) [13], FAN [15] and SOCRATES (Structure-Oriented Cost-Reducing Automatic TESt pattern generation system) [16] are designed to traverse the digital circuit and set logical values at each node of the circuit to sensitize fault propagation to primary outputs (POs). Logic values are set randomly or structural information is used to make an assumption on value assignment. In case of a conflict assignment, the procedure must perform the backtrack operation and try another assignment. These traversing algorithms are very efficient for *easy faults*, where the number of backtracks is low, but usually fail for *hard faults*, where the number of backtracks often exceeds backtrack limits and faults are inaccurately classified as untestable.

Now, let us briefly summarize some advanced techniques and algorithms, which try to enchance the utilization of implicit representations of the vestor set for test patterns generation.

CATAPULT (Concurrent Automatic Testing Allowing Parallelization and Using Limited Topology) [70] generates the whole set of test patterns for a fault by a partial composition of fan-out stem BDDs. First, controllability and observability BDDs are computed by composition/decomposition of BDDs representing fan-out stems and primary inputs. Then, the controllability and observability BDDs are ANDed together. Finally, all stem nodes in the BDD are removed and then the BDDs consist of nodes, which represent primary inputs (PIs) only. Fan-out stem nodes are removed by BDD composition. If a BDD has a path which consists of variables representing PIs only, the fault is testable and the path represents the test vector for a fault. If BDD compositions for all paths fail, the fault is untestable. This approach seems to be promissing for small circuits and *hard faults*, but its application to large circuits can be limited by a great number of compositions/decompositions to be performed.

TSUNAMI [33] is another BDD-based ATPG for the stuck-at fault model. In contrast to CATAPULT [70], TSUNAMI combines the path oriented search used in traditional ATPGs with a BDD-based approach. The faulty and fault-free circuits are traversed to POs and each gate on the path is added to BDD of the faulty and fault-free circuit. Constructed BDDs consist of nodes, which represent PIs only. If the faulty and fault-free BDDs differ, the fault will be propagated (otherwise the algorithm backtracks). Finally, when the PO is reached, the BDD representing test patterns is created by XOR of the faulty and fault-free BDDs. This technique is faster in comparison with CATAPULT, but the final BDD represents only a subset of test vectors detecting a fault at one PO. Moreover, the constructed BDD can explode as in the previous technique. On the other hand, the size of the test set can be dramatically reduced (by up to 70% compared with SOCRATES [16]) by a compaction, which can be performed by ANDing of BDDs representing the test set for each fault (intersection of test sets for faults).

In [71], the time expensive building of BDD is targeted, as it is the main weakness of BDD-based techniques, and algorithms are designed to make a compromise between time and memory consumption. First, the Ordered Binary Decision Diagrams (OBDDs) are generated only for *hard* and *redundant faults*. Test vectors for *easy faults* are generated by traditional path sensitization based test patterns generators (TPGs). Second, the repetition of work for each fault is minimized. It means, that as much computation as possible must be done once in the initialization procedure to simplify further processing. The algorithm for OBDD generation is similar as in the previously mentioned TSUNAMI [33] and suffers from the same problems as previous approaches. The OBDD size can explode and searching for optimal variable ordering (to reduce OBDD size) is a NP-complete problem [56]. An interesting observation is made for variable ordering. The OBDD size can be significantly reduced if the variable representing a fault is on the top (root) of the

OBDD.

The SAT-Based ATPGs [1], [57], [58], [59], [60], [61], [47], [62] described in section 2.6 are another way of test vector generation. Here, the fault is described as an instance of the SAT problem in the Conjunctive Normal Form (CNF). Despite the fact that the SAT problem is NP-complete in general, the instances produced by transformation from the circuit were proved to be easy to solve [34], [A.11]. There is a number of transformations possible, like [1], [43], [44], [45], [46], but the most common one is the Tseitin transformation [1], [45] which produces SAT instances in CNF with the size proportional to the size of the circuit. Moreover, late research on industrial circuits proved that the time of CNF generation often exceeds the solving time (mainly for *easy faults*) [2], [72], [73]. This drawback has been eliminated by the incremental SAT solver which generates CNFs on-the-fly driven by the SAT-solver decision heuristic [2], [72], [74], [75], thus only required parts of CNFs are used (e.g. loaded from repository). Dedicated SAT solvers can benefit from structural information and prune the search space, which makes SAT-based ATPGs extremely efficient for large industrial circuits [59], [60], [61], [47], [62]. The SAT-Based ATPGs can also accommodate multiple-valued logic [76], [77] or different fault models like path delay fault model, transition fault model etc. On the other hand, the output of the SAT solver is only one test vector, while the BDD-based approaches often produce the whole set or a subset of test vectors. Moreover, the test vectors produced by SAT-based ATPGs have a majority of bits specified, which makes the vector compaction very difficult. This drawback was partially overcome by a post-processing strategy introduced in [78], which reduced the number of specified bits by up to 97% (for industrial circuits) and highly increased test vectors compactability. Comprehensive survey on SAT techniques used in SAT-based ATPGs can be found in [2].

In [69], strengths of the BDD and SAT-based techniques are combined to improve the efficiency of the TPG process (speed-up the SAT solver). The efficiency of SAT solvers dramatically depends on the amount of structural information in SAT instances, which is usually very low. This technique [69] introduces the BDD learning scheme which injects conflict clauses into generated SAT instances and speeds up SAT solving. The algorithm traverses the circuit and detects problematical structures like re-convergent fan-outs, and generates their BDDs from characteristic functions of involved gates. Such BDDs represent conflict clauses by paths to the '0' node. These conflict clauses are added to each CNF. The declared run-time reduction is about 60% in comparison with a *"basic ATPG"*. However, its applicability is limited by the BDD size, as in the previous cases. The size of the *problematical structure* must be limited to keep generated BDDs as small as possible.

The SPIRIT algorithm (Satisfiability Problem Implementation for Redundancy Identification and Test generation) [79] is another extension of the SAT-based approach. It preserves the advantages of the SAT-based ATPGs like the unified model, fast and precise implication process, and it minimizes the memory consumption because it reduces duplicities in processed data. The single-cone processing [80] with single path propagation [81], [82] grants a significant memory reduction. Implication graphs and techniques like static [16], recursive [83] and dynamic (unique sensitization, structural dominators) [15], [16], [84], [85] learning, backward justification [82], [86] and others, guarantee efficient and highly robust TPG. The results for benchmark sets ISCAS'85 [87], ISCAS'89 [88], and ITC'99 [89] are comparable with another state-of-the-art SAT-based ATPGs (used in industry) even if techniques like single-cone processing, single path-oriented propagation and backward justification are used very rarely.

3.3 Automatic Test Patterns Generation for Path-Delay Faults

The next suitable application of implicit representations of vector set is in test generation for path delay faults. The algorithm decides whether there exists a cumulative delay fault on a combinational path, which causes the change on the primary outputs (POs) exceed the system clock interval. It means that two successive test vectors are needed to test one path delay fault. The first vector sets initial conditions in the circuit and the second vector triggers transitions propagations along the path to POs. Previous approaches are based on circuit traversing and path sensitization [11], [90], [91], [92] similarly to those used for the stuck-at fault model. New algorithms are rather based on more suitable implicit representations, which can be processed more efficiently (in terms of speed and fault coverage).

The following techniques represent BDD-based and SAT-based approaches to Path Delay Fault (PDF) generation with utilization of implicit representation of the vector set.

BiTeS [93] is a BDD-based ATPG for strong robust path delay faults. For each test path, the algorithm starts from a primary input and adds all side inputs along the path to a BDD. These side inputs are set to non-controlling values (e.g. 1 for AND and 0 for OR) to ensure path sensitization. This BDD represents the whole set of test vectors, thus it is much easier to make the test compaction, like in TSUNAMI [33]. Experimental results confirmed that even such a simple algorithm without preprocessing is faster than traditional path sensitization approaches. However, the BDD can explode for some circuits as in previous BDD-based approaches and in that case the algorithm fails.

MONSOON [21] is a SAT-based ATPG for path delay faults using multiple-valued logic. This ATPG integrates all techniques used in SAT-based ATPGs for stuck-at faults like structural analysis, incremental SAT formulation, multiple-valued logic encoding, etc. This ATPG proved to be more efficient than other state-of-the-art approaches. Moreover, it can handle large hard-to-test industrial circuits with millions of gates.

3.4 Logic Synthesis

Logic synthesis is another field in digital system design which is very attractive for implicit representations. Logic synthesis covers all operations performed between the first logic/functionality description and the final acceptable design, which meets all constraints (size, delay, testability, etc.). An interesting survey on application of BDDs in multi-level logic synthesis can be found in [94]. Two mainstreams of logic synthesis are logic decomposition and logic minimization. The following text briefly describes some interesting research in this area.

3.4.1 Two-Level Minimization

Two-level minimization algorithms have been mostly based on the Quine-McCluskey minimization procedure, which is used to compute minimal sum-of-products. The Quine-McCluskey computation procedure based on BDDs was introduced in [95], but it is efficient only for a small group of problems (solves only 2 out of 20 hard ESPRESSO problems [96]). The base idea is to extract a minimized covering matrix from BDDs of dominance relations. However, BDDs of dominance relations are produced in each step of the computation procedure, which is quite inefficient.

In [49], a new technique for logic minimization is proposed. It is based on BDDs, but it does not require generation of BDDs of dominance relations. Thus it is much more efficient than previous approaches. The two-level minimization problem is transformed into the covering problem and the cyclic core is computed by application of transformations. BDDs are used to represent meta-products of the computation procedure, which is able to efficiently represent a huge number of minterms and prime implicants. This technique is able to solve all ESPRESSO problems [96] as well as all MCNC benchmarks [96]. Moreover, according to [97] it is 10 to more than 100 times faster than previous approaches like [95], [8], [9] and it is able to solve much more complex problems.

SAT-Espresso [98] is two-level logic minimization tool based on improved version of ESPRESSO [8], [9], [99]. This improved version called ESPRESSO-II implements a state-of-the-art heuristic algorithm, which is able to find a nearly minimal cover in a reasonable amount of time for circuits with less than 100 input variables. Minimization of larger functions is possible, but it is very time-consuming because it processes explicitly represented set of prime implicants (cubes). The heuristic algorithm in ESPRESSO-II is based on four functions, EXPAND, IRREDUNDANT, REDUCE, and ESSENTIALS, which are performed in the loop to refine the solution. Runtime analysis of ESPRESSO-II on large benchmarks shows, which functions cause the major performance degradation (particularly, they are REDUCE, ESSENTIALS and IRREDUNDANT). The SAT-Espresso replaces problematic functions by SAT-based checkers, which are able to perform computation much faster and speed up the overall process (mainly for large benchmarks). Yet, even though the SAT-Espresso is faster than ESPRESSO-II, because it introduces SAT based techniques, it is still much less efficient than the previous approach [49].

Previous techniques are often based on searching for prime implicants (all/subset). In [100], searching for prime implicants is addressed by application of integer programming [101], [102]. The problem of prime implicant searching is transformed to minimization integer programming problem. The input Boolean function in CNF is encoded to a set of equations. Then, minimization problem is solved by integer programming. However, the proposed algorithm produces only one prime implicant per call. A set of prime implicants can be produced by algorithm restarting with additional constraints. Unfortunately, the number of prime implicants can grow exponentially with the size of Boolean function. Thus, this technique is not suitable for generation of all prime implicants.

Another technique for prime implicants computation was presented in [103]. The technique is based on integer programming as well as the previous technique [100], but it computes the minimal set of prime implicants of propositional formulas. The search space of the integer programming instance is reduced by an improved generation technique. Moreover, the proposed algorithm benefits from SAT-based techniques like non-chronological backtracking, clause recording procedures and early identification of necessary assignments. Even if the proposed algorithm is more efficient than the previous approach [100], it is unable to handle industrial problems.

In [50], the minimization of Disjoint Sum-Of-Product (DSOP) is proposed. This algorithm simply builds a BDD from the DSOP and uses variable ordering techniques like sifting [104], [105] to keep the BDD size as low as possible. Each path from the root to the 1-terminal represents one DSOP cube. Thus, a reduction of BDD size implicitly causes a reduction of the DSOP size (assuming that the number of paths is proportional to the number of nodes). BDD is able to store a DSOP for large circuits and quickly perform required operations over a cube set. However, the time to build a BDD dramatically depends on efficiency of the variable reordering heuristic, which can fail in some cases.

The previous approach [50] uses BDD reordering for DSOP minimization. A similar technique is introduced in [51] for multi-level logic minimization. However, the author concludes that current heuristics for variable reordering [106], [107] are insufficient and further research is needed to overcome this bottleneck.

3.4.2 Boolean Network Decomposition

The next vital task in logic synthesis is the Boolean and algebraic decomposition. Previous approaches such as Ashenhurst [108] and Curtis[109] process a Boolean function represented by a Karnaugh map, which is very inefficient for larger functions (they explicitly store the on-set, off-set and dc-set). In contrast, the Roth and Karp [110] decomposition uses only on-set and off-set, which can be a little bit more efficient in comparison with Ashenhurst and Curtis, but it still is not sufficient. The nature of decomposition problem suggests that application of implicit representations can be very useful.

One of the first techniques for Boolean decomposition based on an implicit representation of a Boolean function was described in [52]. A Boolean function is represented by a Reduced Ordered Binary Decision Diagram (ROBDD), which is extended to Edge-Value Binary Decision Diagram (EVBDD). EVBDD is exactly the same as ROBDD, except of the value for edge evaluation. The proposed algorithm evaluates edges in EVBDD and computes the bound and free set, which form the cut line in BDD. Then, it is easy to make decomposition on two BDDs (resp. EVBDD). This technique allows making a simple decomposition over BDD, but the BDD size dramatically depends on variable ordering as in previous cases.

BDDs are quite popular, as can be concluded from previous examples. However, they have also disadvantages. They are very suitable for representing on- and off- set of a single output function, but cannot store the don't care-set or handle multi-output logic functions. In [55], the problem of BDD representation for incompletely specified multi-output functions is specified and functional decomposition is addressed. A multi-output Boolean function is represented by a BDD for a characteristic function (BDD_for_CF). Such a BDD consists of nodes representing input and output variables of the combinational circuit. In the case where the node representing a variable is not on the path from the root to the 1-terminal, it is considered to be a Don't Care (DC). The technique for reduction of a Boolean network aims on reduction of the BDD width instead of a simple nodes count reduction. The algorithm recursively performs collapsing of BDD partitions which represent the same function. Furthermore, this reduction is extended by an algorithm, which heuristically searches for minimal clique cover. Suitable parts of BDD are replaced by computed minimal clique covers, which causes reduction in the BDD width. This approach is not applicable for large circuits, because it represents a characteristic function of the whole multi-output Boolean network and the reduction in BDD size introduced by DCs is not sufficient.

The presented techniques are just few examples, which demonstrate possible utilization of implicit representations. BDDs and their modifications are widely used in logic synthesis. However, they are not applicable in cases where the BDD explodes. Techniques like local BDD processing can be very useful for logic synthesis, but they are not applicable in general. Moreover, the size of the local BDD depends significantly on circuit partitioning algorithm.

3.5 Summary

There are a great number of similar tasks and algorithms showing how implicit representations of functions or vector sets can be advantageous. Unfortunately, their complete enumeration is beyond the scope of this work. However, it can be concluded that the most common implicit representations are BDDs (and their modifications), instance of the SAT problem in CNF, or some hybrid approaches (discussed in previous paragraphs). From the summary introduced in this section, a common engineer can conclude that SAT-based approaches are much more efficient than the BDD-based ones. However, this conclusion is not exactly true. It was proved [65] that BDDs and SAT are orthogonal, which means that there are problems where SAT-based techniques are very efficient, while the BDD-based technique fails, and vice versa. Despite of that the SAT-based techniques seem to be much more resistant to failure.

All these examples show that a *suitable* application of an implicit representation for a *suitable* problem can be advantageous and can produce optimal results in a reasonable amount of time. In contrast to the previously mentioned, the application on unsuitable problem can cause performance degradation or produce poor results. This dissertation thesis shows two case studies and discusses possible ways of problem solving by using implicit representations. Moreover, it discusses the suitability of similar problems for application of implicit representations. The implicit representation of the vector set as a

SAT instance in CNF is preferred, because previous research proved, that it is much easier to handle and its application can yield in a great robustness and performance improvement.

Chapter **4**

Properties of Implicitly Represented Vector Set

Previous chapters show that implicit representations of vector sets appearing in logic design and testing have been thoroughly studied for decades. This research can be divided into 3 mainstreams:

1. Circuit-based approach, where the characteristic function of the vector set is described by a combinational circuit and a desired vector is extracted by some circuit traversing algorithm.

2. BDD-based approach, where the characteristic function of the vector set (or the circuit function) is described by a BDD.

3. SAT-based approach, where the characteristic function of the vector set is described by an instance of a SAT problem in CNF. Each assignment of variables, which satisfies the Boolean formula, represents one vector from the set.

This thesis is mainly focused on the vector set representation described as a SAT problem instance in CNF. It is the youngest and a very promising approach, which proved to be suitable in many areas (see chapter 3). The SAT problem is NP-complete in general, but SAT instances produced from combinational circuits are known to be easy to be solved [34], [A.11]. The circuit-to-CNF transformation is fast and the CNF size grows linearly with the circuit size. The circuit representation of the vector set seems to be inefficient for further processing and the BDD representation of the vector set does not seem to be very suitable for big circuits.

This chapter discusses hardness, satisfiability and possibilities of reduction of SAT instances in CNF produced by the Tseitin transformation [1], [45] from a digital circuit. More comprehensive survey can be found in [A.11], [A.5], [A.14], [A.16], [A.13], [A.15].

The difficulty of solving *random* SAT problem instances has been thoroughly studied in the past years. In [111] Selman proposed a metric of difficulty of solving the SAT problem, in terms of the number of the Davis&Putnam [112] algorithm steps. This algorithm has been established as a basis of most of the modern SAT solvers [1], [57], [58], [59], [60], [61], [47], [62]. He has found that there exists a phase transition, where SAT instances rapidly turn from satisfiable to unsatisfiable. At this transition (threshold), D&P-based algorithms suffer from extremely long solving time. Such a threshold was observed at the clauses/variables ratio near 4.3 for 3-SAT, independently of the number of variables. At this threshold, approximately 50% of random instances are satisfiable. An interesting phenomenon is observed for the 2-SAT problem: the SAT/UNSAT transition is continuous, the 50% satisfiability threshold lies near the ratio of 1.0, whereas there is no apparent solving time increase near this threshold. It was shown that 2+p-SAT problems behave like 2-SATs for $p < 0.4$ and like 3-SAT for p higher [39].

By processing approx. 60 circuits from the ISCAS [87], [88] and ITC'99 [89] benchmark sets, we have generated more than 180,000 ATPG SAT instances produced by the Tseitin transformation. By analyzing them we obtained some information on their properties. First, we analyzed the ratios of clauses of a given number of literals. From the nature of the circuit-to-SAT transformation it is expected that 2-literal and 3-literal clauses will prevail. This fact was more or less confirmed. The following Tab. 4.1 shows the average measured percentages of K-literal clauses. The first column shows the clause length, while the second and third column show the average measured percentage of K-literals for SAT instances produced by Tseitin transformation ("Original SATs") and SAT instances reduced by solution preserving reductions ("reduced SATs") described at the end of this section.

Table 4.1: Average percentages of K-literal clauses

Clause length	**Original SATs** [%]	**Reduced SATs** [%]
1-literal	3	0
2-literal	70	86
3-literal	24	11
4-literal	2	2
>4-literal	1	<1

The distribution of 2-literal and 3-literal clauses is illustrated in Fig. 4.1, where 1000 random ATPG SAT instances were tested and numbers of instances evaluated as satisfiable were counted. Judging from this data we can say that SAT instances produced by Tseitin transformation from the digital circuit are similar to 2+p-SATs, where $p = 0.24$ [39]. In order to judge on hardness of these SATs we have computed the clauses/variables ratio, in order to cope with the above-mentioned metrics. The results were surprising: the average ratio was 2.37, ranging from 1.38 to 3.01.

Recall that the satisfiability threshold for 2-SAT or a similar 2+p-SAT is near the ratio of 1.0. I.e., random instances having the above-mentioned clauses/variables ratio should be mostly unsatisfiable, whereas the SATs produced by Tseitin transformation are mostly satisfiable (99% in our measurements), since most of faults in the tested circuits

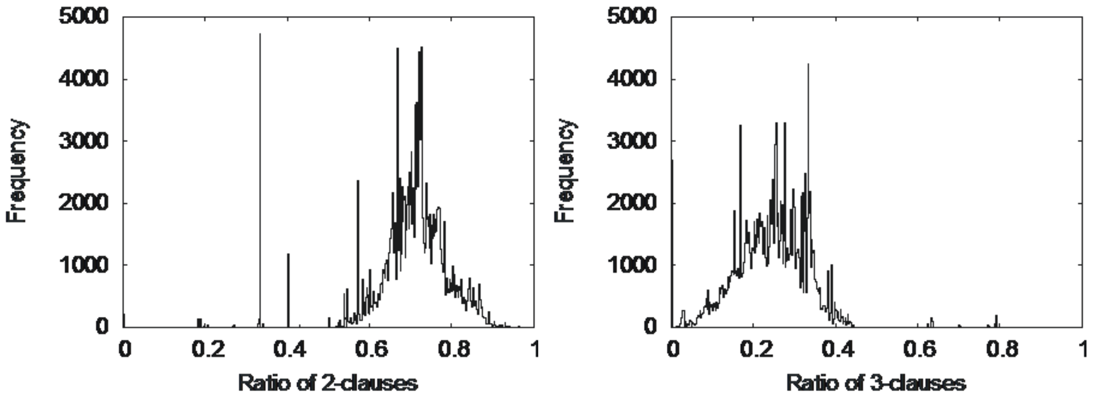

Figure 4.1: Distribution of 2- and 3-literal clauses in ATPG SATs.

are detectable. To justify this surprising fact, we have generated random instances with characteristics exactly like those described in this paragraph, with constantly 300 variables, while we varied the number of clauses. Experimental results have confirmed the theory: all instances having the ratio above 1.8 were unsatisfiable. Here numbers of satisfiable formulas of 1000 random ones were measured. We have also generated random clones of real instances, by exchanging real variables in SAT by random ones. Only 10% of them were satisfiable. This brings us to the first conclusion: ATPG SAT instances produced by the Tseitin transformation do differ from random instances; they are satisfiable, even though they should not be. Up to now, we have no fully relevant clue to explain this phenomenon. We have measured the connectivity of the SAT instances, when expressed as graphs [113]. However, we haven't found a significant difference between SAT instances produced by the Tseitin transformation and random SAT instances of equal parameters.

The second part of our research on properties of SAT instances produced by the Tseitin transformation deals with redundancies in the vector set description and also discusses instance determination to be SAT/UNSAT given by internal constraints.

4.1 Solution Preserving CNF Reductions

Redundant information can be removed from a SAT instance by *solution preserving reductions*. First, we reduce the number of variables. SAT instances often have 1-literal clauses. Variables of these literals must be assigned to a constant value, for the SAT to be satisfied. Repeated application of 1-literal clause elimination will identify most of constantly set variables. The rest of them are detected by fixing the variable to a constant value and solving the SAT problem, see [114]. Next, we reduce the number of clauses by removing duplicities, absorbed clauses, and by creating resolution terms [112]. All these transformations preserve the set of solutions.

Experimental results show that on average 60% of variables and 65% of clauses can be removed this way. After these reductions, the clauses/variables ratio sinks down to 1.67 (from 2.37), which in still in the unsatisfiability region of random SAT instances. The percentage of 2-literal clauses grows up to 86% (from 70%), whereas the percentage of 3-literal clauses sinks to 11% (from 24%), see Tab. 4.1. Notice that the reduced SAT instances are theoretically even simpler to be solved than the unreduced ones, in terms of both the clauses/variables ratio and the percentage of 3-literal clauses.

The next outcome of the reduction was determination of the backbone size [39]. The backbone is a set of variables that are fixed to a constant value for every SAT solution. We have evaluated SAT instance satisfiability using the backbone size [39]. The backbone is removed by the reduction, while its average size was 57% of variables for SAT instances produced by the Tseitin transformation from the combinational circuit. For the randomly generated clones of the benchmarks, the backbone size was 77% of variables. This allows us to state that SAT instances produced by the Tseitin transformation are much less constrained than random ones, and therefore their satisfiability is higher [39].

4.2 Summary

We have made an exploration of the nature of ATPG SAT instances produced by the Tseitin transformation from a digital circuit. We have experimentally evaluated parameters of such SAT instances and found them similar to 2+p-SAT problems, where p = 2.37. This makes these problems very easy to be solved by state-of-the-art SAT solvers. This fully justifies already known facts. However, the easiness of ATPG SAT was formerly proven by analyzing the circuits, instead of the actual SAT instances produced.

Next, we have found that SAT instances produced by the Tseitin transformation significantly differ from randomly generated ones of equal parameters, particularly in terms of their satisfiability. SAT instances produced by Tseitin transformation are mostly satisfiable, even though random instances of the same parameters should not be. We have proposed, and consequently disproved some theories on the reasons in this thesis. Further research is needed to fully understand the satisfiability phenomenon.

Finally, we have found that SAT instances produced by the Tseitin transformation involve a great number of redundancies. On average 60% of variables and 65% of clauses can be removed by application of solution preserving reductions.

Chapter 5

Test Compression

Testing of digital circuits is quite a difficult task, for a huge amount test data needed to be delivered to the circuit under test (CUT). With the growing complexity of designs, scan-based test techniques are becoming a standard. The test patterns are shifted into the chain of scan registers (scan chain) through a serial interface and the circuit under test response is shifted out to the response compactor, see Fig. 5.1.

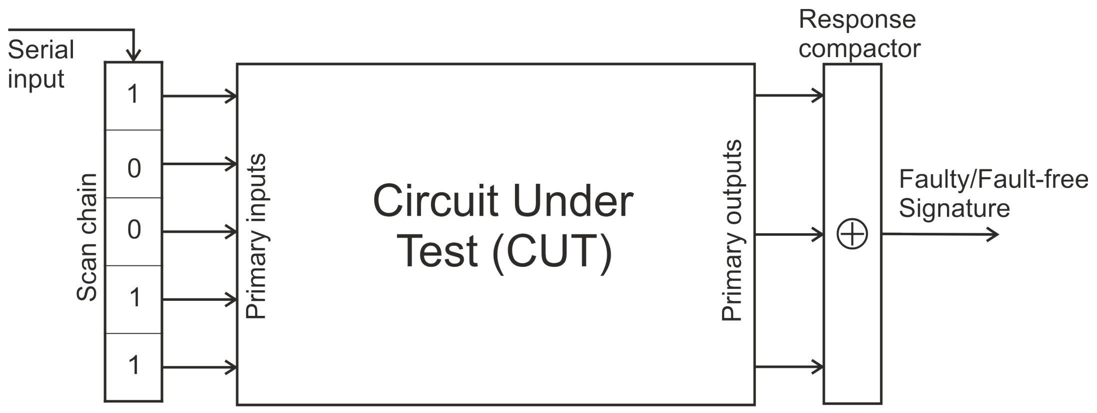

Figure 5.1: Basic scan based test architecture.

The size of such test patterns set grows significantly with the size of the digital circuit and memory consumption becomes unfeasible for large circuits. Thus, the compression of test patterns is a vital task of test patterns generation process. Here, an implicit representation of the test pattern set can significantly reduce memory and time consumption required for test patterns set processing.

Yet, the compression ratio is not the only indicator of test compression applicability. A significant influence on its applicability has the decompression technique and the way of test patterns application on primary inputs of the CUT (Circuit Under Test). In System-on-Chip (SoC) designs compliant with the IEEE P1500 standard [115], [116], the data transfer is realized by a TAM (Test Access Mechanism), which creates an interface between ATE (Automatic Test Equipment) and the on-chip test mechanism. Design requirements force us to make the TAM as narrow as possible, but sending test patterns through a narrow TAM may cause a considerable growth of the testing time. Thus, compressed test patterns are often sent through TAM and the decompression is performed by dedicated hardware on the chip. This approach grants a narrow TAM and keeps the test application time low. However, the complexity of the decompression hardware must be considered.

There are several methods used for test patterns compression. Many of these methods are based on using of some encoding, such as statistical codes [117], [118], [119], [120], [121], run-length codes [118], [119], [120], [122], [123], and Golomb codes [124], others are based on XOR networks [125], [126], hybrid patterns [127], EDT (Embedded Deterministic Test) [128] and reuse of scan chains [5].

This chapter deals with the first pilot example which is focused on the test patterns compression based on patterns overlapping [5], [6], [7], [30] (see subsection 5.1.2 for more details). Previous compression techniques [5], [6], [7], [30], which are based on test patterns overlapping try to compress test patterns pre-generated by an ATPG. Pre-generated test patterns are produced blindly regardless of their suitability for pattern overlapping, which can significantly influence the algorithm efficiency (the compression ratio). Moreover, the size of pre-generated test patterns set grows significantly with the size of the circuit, and their storing or direct processing can be unfeasible. Here, some implicit representation of the test patterns becomes very handy. The set of all test patterns for each fault can be represented implicitly and a suitable test pattern or their subset is produced by application of additional constraints. These additional constraints reduce the set of test patterns in an informed way, e.g., to grant maximal overlap of test patterns, reduce the switching activity in the scan chain, etc. The basic principles of this Constraint Test Pattern Generation (CTPG) are shown in Fig. 5.2.

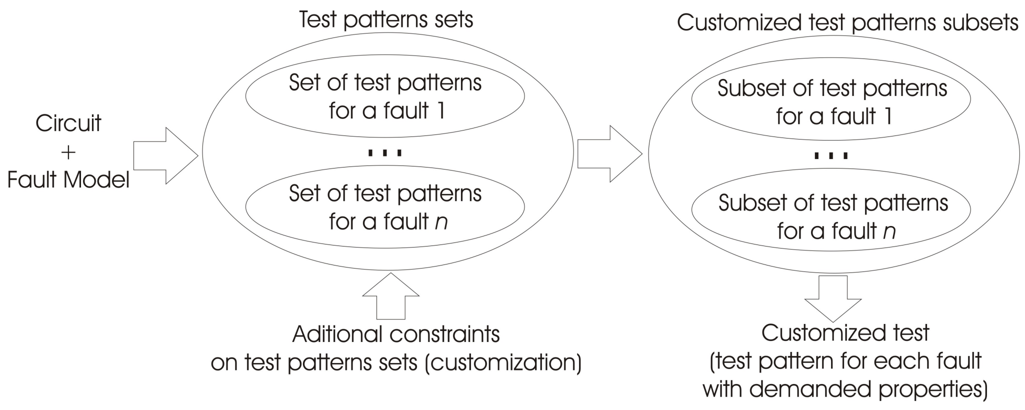
Test patterns sets
Customized test patterns subsets
Circuit + Fault Model
Set of test patterns for a fault 1
...
Set of test patterns for a fault n
Subset of test patterns for a fault 1
...
Subset of test patterns for a fault n
Aditional constraints on test patterns sets (customization)
Customized test (test pattern for each fault with demanded properties)

Figure 5.2: Basic concept of the constrained test generation.

This thesis introduces a novel test patterns compression algorithm *SAT-Compress* [A.10], [A.9], [A.17], [A.5] based on a design of a dedicated SAT-ATPG [1], [2] and CTPG principles. Unlike in previous approaches, the test patterns are generated on the fly to reach the locally best overlap and maximize the compression. Test patterns compressed this way can be easily decompressed by the RESPIN (REusing Scan chains for test Pattern decompression) decompression architecture [30], which is intended to test system on chip (SoC) cores by reusing scan chains.

Generally, this research is focused on a class of algorithms where the same set of SAT instances is repeatedly processed with different constraints. Algorithms are forced to handle repeated processing of the same SAT instances in CNF, which can cause a significant time overhead [A.10]. It is expected, that the majority of constrained SAT instances are classified as UNSAT (UNSATisfiable) [A.11], [A.10]. It means that they do not contribute by any sub-solution and must be repeatedly solved with other constraints.
Our experiments on CNF processing show the differences (time/memory consumption) between

- their repeated generation,
- storing, and
- storing of reduced CNFs.

Reduction of stored CNFs is made by solution set preserving SAT transformations [A.11], e.g., by resolution or propagation of 1-literal clauses. These reductions do not change the set of solutions.

Next, possibilities of detecting unsatisfiable constrained CNFs were explored. The UNSAT instances are early detected by resolving conflicts between the demanded fixed values of the signals in the circuit and their values obtained by CNF implications. The SAT solving of these unsatisfiable instances can be skipped, which can significantly speed up the algorithm. Such a process is in this thesis referred as *static* or *dynamic* UNSAT filter based on the performed type of implications.

Another issue in test patterns compression based on patterns overlapping is the number of bits specified (care)/unspecified (don't care). Specified (care) bits in a test pattern produce constraints for the subsequent test patterns, while unspecified (don't care) bits can be assigned any value. Thus, don't care values introduce a kind of flexibility in pattern overlapping. It is supposed, that the number of don't cares should be as high as possible to ensure high compression ratio [6], [35], [36].

For this purpose we extend the SAT-Compress algorithm [A.10], [A.5] by injection of "don't cares" into test cubes. The SAT Compress ATPG algorithm generates the compressed test stream by constraining a conventional SAT-based ATPG. Conventional SAT solvers [129], [130], [131] used as the vital part of most of SAT-based ATPG tools produce completely specified solutions (all variables are assigned a value in the satisfying solution). There are several ways of introducing don't cares (unassigned variables) into the SAT solution. First, there are SAT-solvers producing incompletely specified cubes directly [103], [132], [133], [134]. Here satisfying solutions compromising minimum literals (minimal models, prime implicants) are generated. However, the optimization criterion is computed over all variables, which is unsuitable for applications, where values of only some variables (circuit primary inputs) are of interest.

Next, optimization version of SAT can be transformed to Integer Linear Programming (ILP) [100]. Here the optimization criterion can be modified for our purposes, so that only some variables are accounted in its computation. We introduce a similar method, particularly the conversion of the SAT problem minimizing the number of assigned variables in the satisfying solution to Pseudo-Boolean Optimization (PBO) [135].

Finally, don't cares can be injected into a completely specified vector obtained from a conventional SAT solver [129], [130], [131], while the coverage is checked by simulation. When fault simulation is performed, we get additional information on the obtained test cube - its fault coverage [A.7]. Then we can, e.g., inject don't cares while respecting the fault coverage. This is the informed way of obtaining test don't cares proposed in this thesis. We compare this simulation-based method with the uninformed ones and show its benefits in test compression, in terms of both the compressed test stream size and test compression time.

Advantages and disadvantages of these approaches are discussed over the results and recommendations for further design of SAT-based constrained test patterns generation algorithms are proposed.

5.1 State-Of-The-Art

This chapter briefly summarizes the state-of-the-art in the field of constrained test patterns generation (CTPG) and its special case, a serial compression based on a test patterns overlapping.

5.1.1 Constrained Test Patterns Generation

Generation of test patterns with some constraints imposed is a common process in digital circuits testing. Test patterns can be constrained for various purposes:

- to be better compressed [5], [A.10],
- to limit the SAT solver search space,
- to exclude invalid input combinations, etc. [1], [32], [136], [137], [138].

One example application of a constrained ATPG is a broadside transition testing [137]. A conventional ATPG based on PODEM [13] algorithm produces a set of test patterns with a significant number of patterns covering functionally untestable transition faults. These functionally untestable transition faults need not be tested because they do not affect the normal functionality of the chip (errors caused by these faults cannot occur). Nevertheless, testing the chip for these faults may cause the test fail, and thus decrease the yield. Thus, an ATPG is constrained by a set of forbidden variable assignments that enable detection of the functionally untestable transition faults. These constraints are described by a Boolean formula in CNF. The constrained ATPG fixes variables in the generated test pattern and at the same time it fixes the corresponding variables in the CNF of constraints and checks the CNF for conflicts in the variable assignment. When a conflict occurs, the ATPG backtracks and searches for a different variable assignment in the test pattern. The test set generated this way does not activate functionally untestable transitions, which increases the quality of the test and reduces the yield loss caused by testing of the functionally untestable faults.

In [136], constraining of test patterns to generate them by cellular automata is presented. All test sequences for a fault are checked for conflicts with rule matrices of cellular automata. The entire set of the test sequences for the circuit under test is implicitly represented by a BDD (Binary Decision Diagram) [25]. The BDD is used to select only those sequences which can be reproduced by cellular automata.

An ATPG for industrial circuits with restrictors [138] represents another application of constrained test patterns generation. Industrial circuits contain a great number of buses, tri-state elements and other parts, where the set of permitted signal values is restricted. This structural information is stored as a set of restrictions, which are used by an ATPG to prune the search space and speed up the test patterns generation. This method was implemented as a conventional FAN algorithm [15] extended by a concept of restrictors (constraints).

Low power tests are mostly built from pre-generated test patterns [32] by their reordering, to decrease the dynamic power consumption (i.e., the switching activity) during the test. However, constraints can also be formed to guide the test patterns generation process, in order to generate low power tests directly [32].

The constrained test patterns generation principle may be efficiently employed to compress test patterns. In [5], test patterns are customized for testing the circuit using the RESPIN (REusing Scan chains for test Pattern decompression) architecture [139], [140]. This architecture is targeted to systems on chip (SoCs). To update values stored in the scan chain of the core under test, scan chains belonging to other cores are used. Test patterns are compressed by overlapping [141]. Suitable test patterns are produced by a conventional ATPG tool performing dynamic compaction [142]. Constraints to the circuit's primary inputs are applied, in order to reach a locally optimum overlap with the vector already present in the scan chain from the previous test cycle.

The latest approach to transition delay faults (TDF) [143] test compression is based on a constrained SAT solving, too [144]. TDF can be detected by a pair of test patterns applied in two subsequent clock cycles. Test compression is performed by test patterns overlapping as in the SAT-Compress algorithm, but pairs of test patterns are overlapped instead.

The SAT-based CTPG algorithm can also be enhanced by techniques used in common SAT-based ATPGs. These techniques can be divided into two groups. The first group of techniques deals with SAT solver acceleration by, e.g., variable ordering for the SAT solver heuristic [145], circuit-based dynamic learning [146] or different clause learning techniques [69]. The second group of techniques deals with the reduction of the time overhead caused by the CNF generation e.g. dynamic clause activation [146].

5.1.2 Serial Test Patterns Compression

The serial test patterns compression technique is basically based on finding the best overlap of test patterns, which are usually pre-generated by an ATPG. An illustrative example of this overlapping-based compression is shown in Fig. 5.3. Here the non-compressed test length equals to the number of patterns multiplied by the number of CUT scan-chain cells, 10x5 = 50 bits in the example case. When properly overlapped, the compressed test length is 16 bits only.

Note that by shifting the pattern by one bit only, the overlap needs not be always achieved. Then two or more clock cycles (shifts) must be applied. Such a case is in [140] referred to as a presence of *link patterns.* They do not increase the fault coverage, but may increase the defect coverage. In Fig. 5.3 there are link patterns in the 6th and 11th clock cycle.

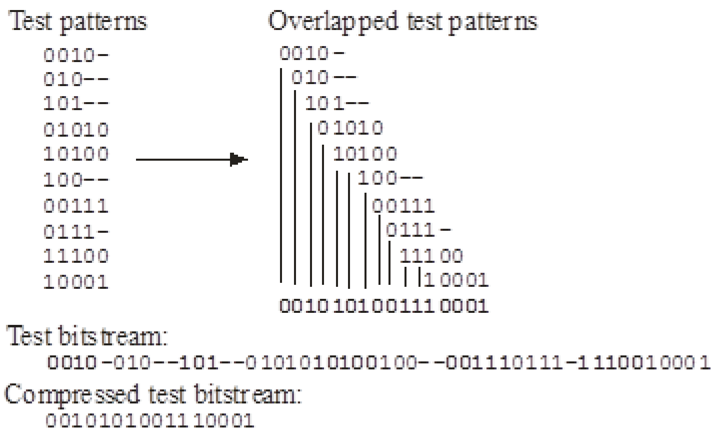

Figure 5.3: Patterns overlapping.

The serial compression technique was described in [141] for the first time. This algorithm generally tries to find contiguous and consecutive test patterns having the maximum overlap. Deterministic test patterns are generated by an ATPG and compacted. Patterns in the scan chain are checked whether they match with one or more test patterns which were not employed in the sequence yet. In [7], the pattern overlapping problem is converted into a Traveling Salesman Problem (TSP), for which different heuristics have been proposed.

The COMPAS (Compressed Test Pattern Sequencer for Scan Based Circuits) [6] test patterns compression tool is based on a similar approach, but it does not use compacted test patterns. Test patterns that are to be compressed are pre-generated by an ATPG as well. However, these test patterns should contain as many don't care bits as possible. For this purpose, one test pattern for each fault is generated. Greater number of don't care bits grants the algorithm much more possibilities to combine test patterns and reach better compression. Other improvements are the simulation after every test pattern application and searching for best successors of a given starting pattern (usually an all-zero pattern). These improvements make COMPAS very efficient in comparison with other compression tools. One of the weakness of COMPAS is the need for don't care bits. The number of DCs and the fact that the algorithm fully relies on a pre-generated test set can also affect the test compression ratio. When the test patterns are highly specified (they contain only few don't cares), it is much harder for COMPAS to find a good overlap of the test patterns and the efficiency of the test patterns encoding decreases, which causes greater memory consumption [35].

Another compression technique is presented in [5] (see next chapters for more details). This technique is based on a *tailoring of test patterns* for a scan-chain. Suitable test patterns are produced by a standard ATPG tool performing dynamic compaction, while constraints to circuit inputs are applied.

The state-of-the-art compression/decompression architecture used in industry is the Embedded Deterministic Test (EDT) [128]. Here a high test compression is achieved by employing a dedicated but generic test decompressor. The compressed test patterns serve as seeds for a pseudo-random pattern generator ("ring register"), where they are decompressed and further distributed to scan-chains using a XOR-network structure ("phase-shifter"). The compressed test patterns are obtained as a solution of a set of linear equations. Similarly to the previously mentioned test compression methods, the test patterns to be compressed need to be pre-computed by an ATPG. Again, high amount of test don't cares is essential for achieving a good compression ratio [128].

5.1.3 The RESPIN Architecture

The SAT-Compress algorithm [A.10], [A.5] and also its enhancements proposed in this thesis are based on the RESPIN (REusing Scan chains for test Pattern decompression) architecture [140], which is targeted to System-on-Chip (SoC) designs compliant with the IEEE P1500 standard [115], [116]. Only a very small modification of P1500 (addition of one multiplexer) can accomplish the test decompression job.

The RESPIN decompression architecture is very suitable for compression techniques based on test patterns overlapping like COMPAS. This architecture is based on full scan, thus only combinational logic is tested. Therefore, the order of patterns in which they are applied to the CUT is *insignificant.* The patterns may then be *reordered*, to reach maximum compression (i.e., maximum overlap).

Further, as mentioned above, standard circuit-based ATPGs [14], [12], [13], [15], [16], [147] are able to generate test cubes with a huge amount of don't care values. Test don't cares are greatly beneficial for the compression, since they can be overlapped with any value (see Fig. 5.3). Thus, don't cares bring more freedom into the overlapping process. These two principles are aimed to be fully exploited by RESPIN-based compression techniques [5], [6], [A.10], [A.5], [140], [148], [149].

The basic idea of RESPIN is illustrated in Fig. 5.4. Multiple embedded cores are considered here. To test one core (CUT - Core under Test), the test decmpression is performed by another core (ETC - Embedded Tester Core).

RESPIN uses two features of P1500 - the serial and parallel test access modes. The compressed test bitstream serially enters the ETC, which is configured as a shift-register. Then the decompressed data is applied to the CUT, which is tested in the parallel scan-chain mode.

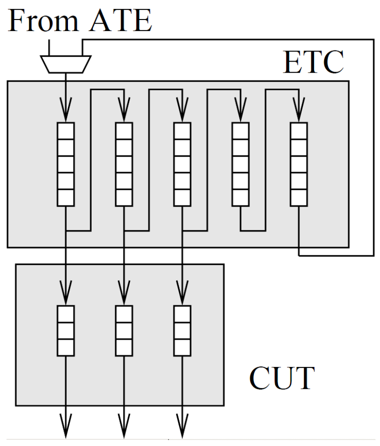

Figure 5.4: RESPIN architecture [140].

The ETC is provided with a multiplexer, enabling *rotation* of the pattern. Thereby, if no data come from the ATE, no information on the stored pattern is lost. This opens a simple way to compression: when the deterministic non compressed test patterns overlap when *rotated* by k bits, each test pattern to be applied to the CUT involves only k (or less) bits coming from the ATE. Actually, rotation needs not be used in practice. In practice, one bit comes from ATE in each cycle, while the remaining bits of the pattern are formed by shifting the previous pattern by one bit. This approach eliminates the need for any control data provided by ATE. For details see [140].

5.1.4 The RESPIN Algorithm

In order to show the main differences between the original RESPIN compression algorithm [140] and SAT-Compress [A.10], [A.5], which is proposed in this thesis, we will briefly review the RESPIN algorithm. The SAT-Compress algorithm is described in section 5.2.

The RESPIN algorithm (see Fig. 5.5) starts with running a conventional ATPG to produce a test T for the circuit (1). Then the constraints cube c is initialized with the initial test vector (2). This can be the previous content of the ETC scan-chain, an all-zero pattern, or it can be decided by the compression algorithm. For the latter case, c is initialized to an all-DC cube.

Then all test cubes from T are tried for merging with c (5). If it is possible (there is a non-empty intersection of the cubes), c is constrained by T (6) and T is removed from the test set (7).

When all test vectors have been tried for merging, a new bitstream bit is formed (10), the constraint cube is shifted by one bit (11), and the released position is assigned a don't care (12). Here n represents the number of circuit primary inputs. The process repeats, until the test set is not empty (3). Then the bitstream is completed by the remaining bits in c (14).

Note that if the merging (4, 5) fails for all vectors from T, a new bitstream bit is formed anyway; the link pattern is thus generated (see subsection 5.1.1).

Such an approach is apparently greedy, and is crucially influenced by the pre-generated test T. Don't care values in the test enable easier cube merging (5), thus higher compression.

```
RESPIN (circuit)
1  Generate test T for circuit
2  c = tp_0
3  while (T ≠ ∅) {
4      for each t ∈ T {
5          if (c ∩ t  ≠ ∅) {
6               c = c ∩ t
7               T = T - t
8          }
9      }
10     bitstream += c[0]
11     c[0...n-2] = c[1...n-1]
12     c[n-1] = DC
13 }
14 bitstream += c[0...n-2]
15 return bitstream
```

Figure 5.5: The RESPIN algorithm.

5.2 The SAT-Compress Algorithm

In our SAT-Compress algorithm we try to eliminate weaknesses of previous approaches [5], [6], [7]. Each fault has its set of test patterns by which it is detected. If we were able to pick the right pattern for each fault in the right order, we could have reached the best possible compressed bitstream for a given fault list. Since explicit computation and storing of all these test patterns is inefficient (and mostly even infeasible), we were forced to find another, more efficient way of test patterns set representation and processing - using an implicit representation of test patterns.

First, we try to avoid limitations on test patterns compression given by pregenerated test patterns (one test pattern for a fault) which are used in previous approaches [6], [7], [128], [150], [151], [152], [140], [148], [149]. The SAT-Compress algorithm does not rely on pre-generated test patterns. The most suitable test patterns are generated on the fly, to reach the (locally) best overlap. However, the locally best overlap of the test patterns does not guarantee the shortest length of the generated bitstream.

```
SAT-Compress ($circuit$)
Generate $FL$ for circuit
$FL$ = $FL$ - Redundant_faults
$c$ = $tp_0$
while ($FL \neq \emptyset$) {
    for each $f \in FL$ {
        $\varphi$ = Create_CNF($circuit$, $f$)
        $\varphi$ = Apply_constraints($c$, $\varphi$)
        $s$ = SAT($\varphi$)
        if ($s \neq \emptyset$) {
            $s$ = Assignment_of_PIs($s$)
            break
        }
    }
    $bitstream$ += $s$[0]
    $FL$ = $FL$ - Detected_by_simulation($s$)
    $c$[0...$n$-2] = $c$[1...$n$-1]
    $c$[$n$-1] = DC
}
$bitstream$ += $c$[0...$n$-2]
return $bitstream$
```

Figure 5.6: The SAT-Compress algorithm.

Further, we have researched properties of implicit representations of test patterns. We have found that we can take advantage of principles of SAT-based ATPGs and efficiently represent all test patterns for each fault implicitly, by one SAT problem instance in a CNF. The CNF test set representation is much less memory consuming than a standard tabular test set representation.

The size of the SAT instance is linear with the circuit size, therefore such an approach imposes no big computational and memory overhead. For details of the circuit to CNF conversion, see [1], [2].

SAT-Compress uses an implicit representation of all test patterns for a given fault as a SAT instance described in a conjunctive normal norm (CNF). Any satisfying solution of the related CNF-SAT problem represents a test vector for the fault [1] and vice versa. If the CNF is not satisfiable, the fault is undetectable (redundant).

The SAT-Compress algorithm is targeted to the RESPIN architecture. The compressed test is produced by constraining SAT instances by patterns stored in the ETC (see subsection 5.1.3 and subsection 5.1.4). In the RESPIN algorithm, a conventional (commercial) ATPG was constrained in a similar way. However, SAT based representation of test vectors offers much higher flexibility and possibilities of to reach much higher compression ratios [A.5].

Similarly to RESPIN and COMPAS, SAT-Compress tries to find the best overlap of test patterns by gradually building the compressed test bitstream, while each generated test pattern imposes constraints on the subsequent test patterns. The basic algorithm is shown in Fig. 5.6.

First, a fault list for the circuit is generated (1), from which redundant faults are removed (2). The constraint cube is set to the initial test pattern, as in RESPIN (3). The compressed bitstream is gradually constructed in the main loop of the algorithm (4-18), fault by fault (5). A CNF is generated for the processed fault (6), constraints in form of unit clauses are applied to this CNF (7), and SAT is solved (8). If the constrained formula φ is satisfiable, constraints are refined by the assignment of primary inputs (PIs) in the SAT solution (10). Then the loop is terminated (11). Note that the new constraint cube is a subcube of the former one, since the original constraints are included in the SAT instance (7).

Then a new bitstream bit is formed (14), faults detected by the pattern are removed from the fault-list (15), and new constraints are formed by a shift left (16-17). The test generation continues until the fault list is not empty (4). Finally, the bitstream is completed by the bits remaining in c (19).

Note that compared to RESPIN and similar algorithms, there is no concept of test set in SAT-Compress; the test is represented implicitly. On the other hand, RESPIN does not operate with concepts as fault and circuit - these are treated in the ATPG phase run prior to the compression algorithm.

5.2.1 Experimental results

This subsection summarizes experimental results of SAT-Compress (see Fig. 5.6) and makes a comparison with other state-of-the-art compression techniques. A brief comparison of data volume reduction for state-of-the-art compression techniques is shown in Tab. 5.1. The first column *"Circ."* represents the name of the benchmark circuit. A comparison for only seven biggest ISCAS'89 circuits [88] is shown, since no more relevant data for the other methods were available to us. The compressed test lengths in bits, for nine different competitive methods, are shown next. The last column shows the compressed test data size in bits for the SAT-Compress compression tool. An all-zero initial test pattern for both COMPAS and SAT-Compress is used, thus the results are not influenced by different initial states.

The measurement was performed on Intel Xenon CPU - 2GHz with 4GB RAM.

It can be concluded from Tab. 5.1, that the SAT-Compress algorithm can reach similar and often even better compression of test patterns than most of the presented state-of-the-art compression methods. SAT-Compress can theoretically reach better results, if a more efficient fault-processing heuristic was used. Currently, the faults are processed in a greedy, first-only way. Next chapters show simple extensions of the SAT-compress algorithm like fault filtering or a more robust don't care processing.

Table 5.1: Comparison of the test data volume for different compression techniques

Circ.	MinTest [153]	Stat. Coding [121]	LFSR Reseeding [117]	Illinois Scan [152]	FDR Codes [122], [154]	EDT [128]	RESPIN++ [30]	COMPAS [6]	SAT-Compress
s5378	20758	15417	6180	14572	12346	-	17332	2148	2407
s9234	25935	19912	12112	27111	22152	-	17198	11594	9928
s13207	163100	52741	11285	109772	30880	10585	26004	4163	10457
s15850	58656	49163	12438	32758	26000	9805	32226	8234	12987
s35932	21156	-	-	-	22744	-	-	1860	5096
s38417	113152	172216	34767	96269	93466	31458	89132	24198	19291
s38584	161040	128046	29397	96056	77812	18568	63232	7291	14271

The result quality of most EDA processes based on local heuristics depends on random aspects coming from the input description [155], [A.3], [A.4]. To diminish the influence of randomness on evaluation, most experiments were conducted repeatedly, with random initial patterns and random faults ordering, and the results were averaged.

As stated above, there are many random aspects that influence the test generation process. First of all, it is the selection of the initial test pattern (tp0 in Fig. 5.6). It defines the initial constraints and therefore it influences the whole run of the greedy algorithm. The same holds for the ordering of the fault list; the fault list is traversed sequentially until a test vector detecting some fault is found (see Fig. 5.6, step 5). Different orderings of the fault list will induce different runs of the test generation heuristic. Also the order in which don't cares are injected (see Fig. 5.15, line 3) influences the final bitstream length as well as permutation of primary inputs.

The time consumed by the presented tools could not be compared because neither their source codes, nor executables were available. Nevertheless, the time-consumption of the SAT-Compress algorithm is shown in Tab. 5.2, which compares lengths of compressed bitstreams for the SAT-Compress and COMPAS as representatives of serial test patterns compression techniques.

The first column *"Circ."* presents the name of the benchmark circuit. The compressed test lengths generated by COMPAS and SAT-Compress, both starting with an all-zero test pattern, are shown in the column *"Bits"*. Then we have repeatedly run the algorithms starting with different initial test patterns. The average compressed test bitstream lengths are shown in the columns *"Avg."* and average variations of compressed test patterns lengths are shown in the columns *"Var."*. The time consumptions of test patterns compression for SAT-Compress in seconds are shown in the *"Time"* column. The results of this measurement show that the bitstream length for both tools significantly depends on the initial test pattern and starting with an all-zero seed (which is the default setting for COMPAS) can produce outstandingly poor results, out of the range of the variation (see, e.g., s298 for COMPAS). In some cases, our tool reaches much better compression than COMPAS, but it may also fail. For an unknown reason the efficiency of the proposed algorithm is much better for ISCAS'85 benchmarks than for the ISCAS'89 ones. As can be seen from Tab. 5.2, the time consumption may be considerable for larger circuits, but we suppose, that scalability of the proposed algorithm may be improved by using more scan chains and division of the circuit into smaller parts.

Then, we have evaluated random aspects which can affect the compressed bitstream length. For greedy algorithms like the SAT-Compress and COMPAS, the compressed bitrstream length can be most affected by initial conditions. In this context, the compression ratio can be influenced by the initial test pattern, which is loaded in the scan-chain and the order of circuit's primary inputs in the scan-chain (permutation of PIs). Both these initial conditions are given by hardware design, thus their optimization is limited.

First, we have repeatedly run the algorithms starting with different initial test patterns. Fig. 5.7, Fig. 5.8, and Fig. 5.9 show examples of distributions of bitstream lengths using different starting patterns for COMPAS and SAT-Compress. In both cases we always start with a test pattern covering one particular fault. The number of restarts is equal to the

Table 5.2: COMPAS and SAT-Compress test lengths and run-times

Circ.	COMPAS			SAT-Compress			
	Bits	Avg.	Var.	Bits	Avg.	Var.	Time [s]
c17	9	9	1.5	11	12.3	1	0
c432	195	218	30.6	189	198.2	21.8	3
c499	260	303.7	47.6	187	198.2	8.7	6
c880	540	412.7	44.3	410	561.5	60.99	10
c1355	1040	1126	68.6	349	-	-	74
c1908	1009	989.8	49.6	624	-	-	229
c2670	6553	5940	269.7	2223	-	-	2305
c3540	747	743.7	32.9	1622	-	-	3569
c5315	1255	1159.5	64.8	881	-	-	156
c6288	82	95.2	36.6	97	-	-	165
c7552	6005	6430.7	345.2	4840	-	-	7406
s27	16	13.2	2.3	23	21.5	2.1	0
s208	130	124.1	9.1	202	209	15.9	0
s298	101	79.6	5.5	137	139.7	8.2	0
s344	85	80.9	6.7	116	114.7	10.2	0
s349	85	80.1	6.8	116	114.4	10	0
s382	123	111.3	6.6	191	179.2	12.8	0
s386	264	255	10.8	304	319	12.5	4
s400	121	109	7.2	173	170.9	9.7	0
s420	352	315.9	29.1	624	574.8	46.4	11
s444	116	107	7.5	160	150.6	10.5	0
s510	160	156	8.9	210	211.8	10.3	2
s526	344	349.8	18.4	521	482.8	22.7	4
s526n	344	350.2	18.1	493	-	-	3
s641	397	393.3	24.2	710	670.7	28.4	15
s713	428	403.8	26.1	642	672	38.7	32
s820	460	504.6	21.9	697	697.7	26.3	28
s832	494	498.8	19.5	729	690.7	24.1	36
s838	920	762.3	79.6	1800	1638.3	103.5	136
s953	723	700.4	24.7	825	-	-	51
s1196	740	738.5	23.5	1211	1202.2	41.9	173
s1238	741	769.2	24.3	1300	1268.7	41.7	365
s1423	596	621.5	38.6	794	827.6	43.9	72
s1488	488	461.5	15.1	546	-	-	20
s1494	431	451.3	16.6	573	-	-	25
s5378	2148	1995.6	73.8	2407	-	-	631
s9234	11594	11309.6	310.7	9928	-	-	68119
s13207	4163	-	-	10457	-	-	67751
s15850	8234	-	-	12987	-	-	114932
s38417	24198	24926.3	1717.7	19291	-	-	91713
s38584	7291	-	-	14271	-	-	122143

number of faults.

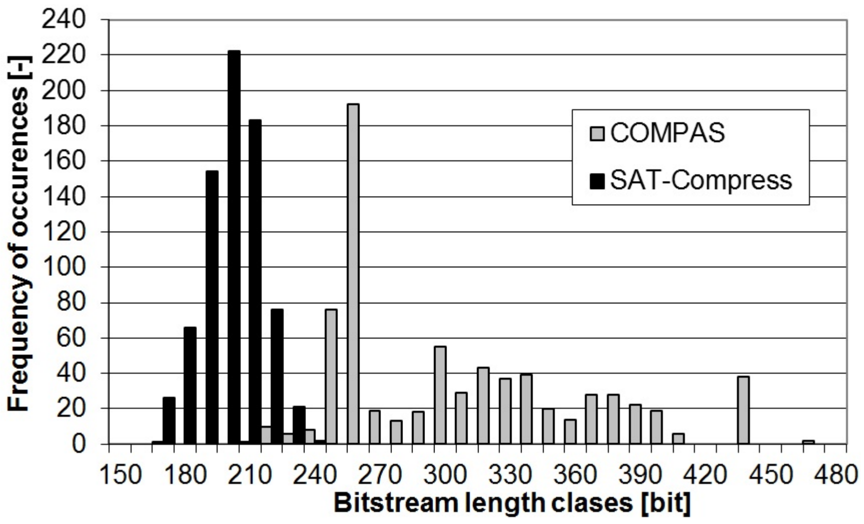

Figure 5.7: Frequency of bitstream length distribution for different initial test patterns (c499).

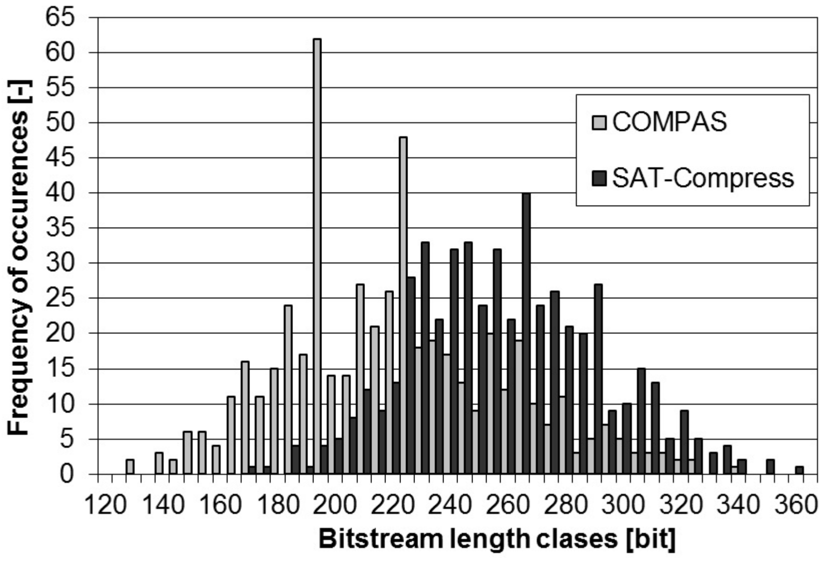

Figure 5.8: Frequency of bitstream length distribution for different initial test patterns (c432).

As can be seen, the compressed bitstream length seems to have Gaussian-like distribution and the only difference between SAT-Compress and COMPAS characteristics is their

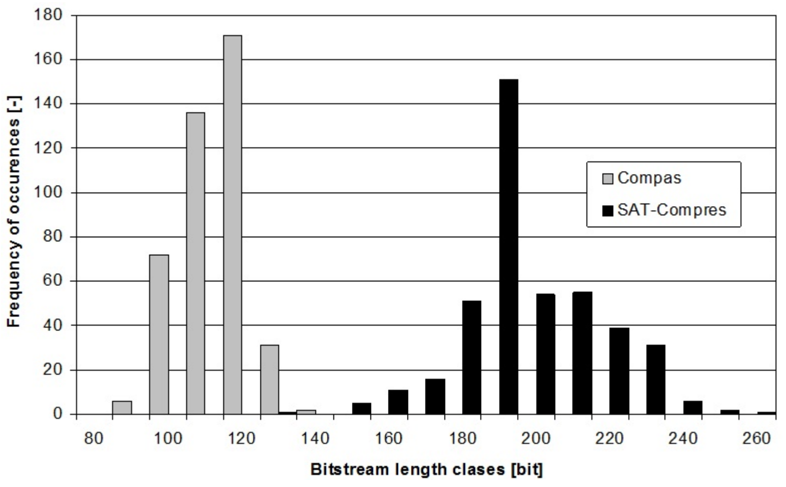

Figure 5.9: Frequency of bitstream length distribution for different initial test patterns (s400).

displacement. This Gaussian distribution of the compressed bitstream lengths for different starting seeds is a common characteristic of these two tools for all tested benchmarks. It can be seen that the selection of the initial test pattern has a crucial impact on the resulting compressed test length, for both algorithms.

Then, we have evaluated the influence of permutation of PIs in scan-chains on the length of the compressed bitstream. This measurement has been performed for the SAT-Compress only (we were unable to perform measuerement for COMPAS) and the influence of PIs permutation on bitstream length is compared with the influence of initial test pattern on the bitstream length. Results are shown in Fig. 5.10, Fig. 5.11 and Fig. 5.12. As can be seen, the compressed bitstream length seems to have a Gaussian-like distribution and their displacement indicates that the permutation of PIs in the scan-chain influences the compression ratio more than the initial pattern.

This chapter shows that the efficiency of a simple algorithm such as the SAT-Compress can be very high. In section 5.3, we mentioned that the main weakness of previous approaches is the pre-generated set of test patterns and the number of don't care bits in these test patterns. However, Tab. 5.2 as well as Fig. 5.7, Fig. 5.8 and Fig. 5.9 show that the compression ratio significantly depends on the initial test pattern. Thus, following chapters study other influences on serial compression performed by the SAT-Compress algorithm in more detail. Experimental results are discussed to show effects which influence the compression ratio and performance most significantly. Finally, the benefits given by application of implicit representation is discussed as well as its applicability to similar problems.

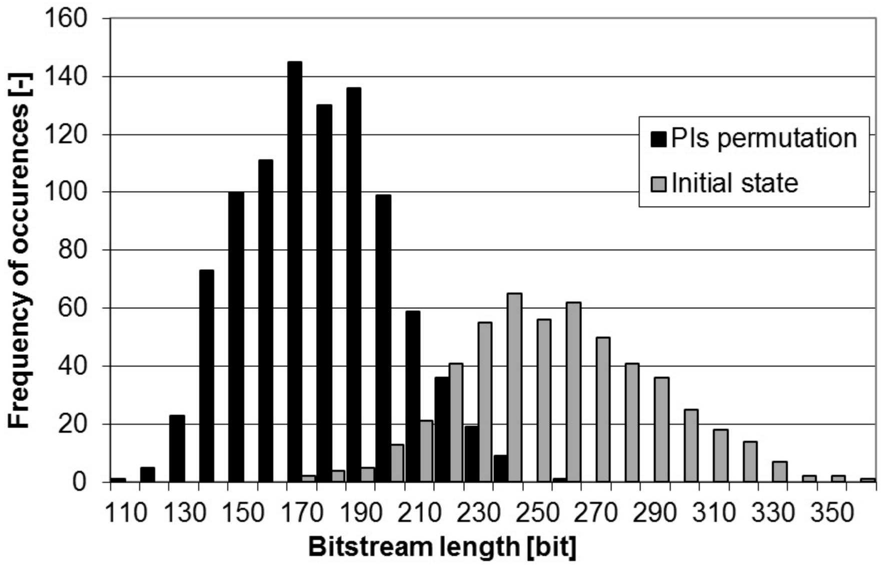

Figure 5.10: The influence of different initial test pattern and PIs permutation on bitstream length distribution for the SAT-Compress algorithm (c432).

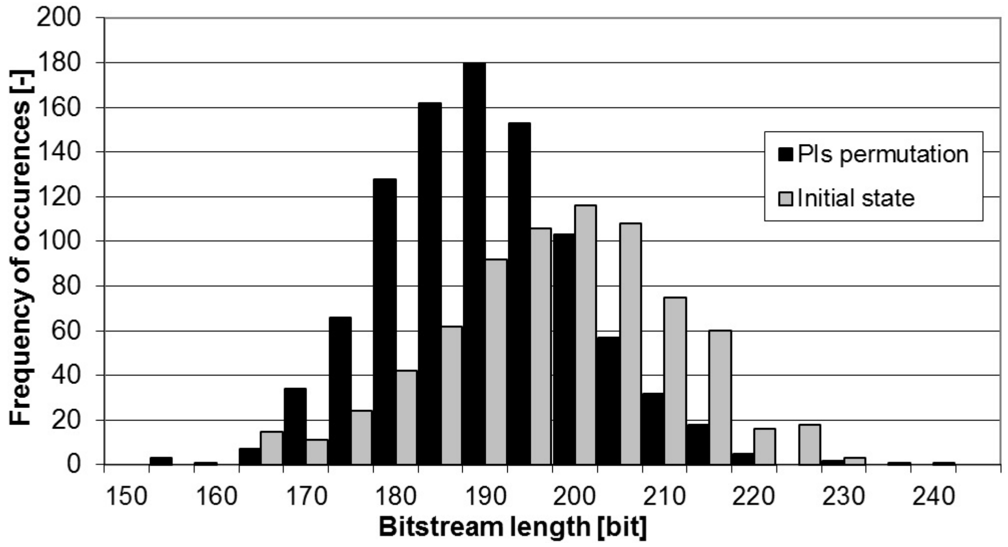

Figure 5.11: The influence of different initial test pattern and PIs permutation on bitstream length distribution for the SAT-Compress algorithm (c499).

5.3 Obtaining Test Don't Cares

If a conventional SAT solver ([129], [130], [131]) is used in step (8) of the algorithm in Fig. 5.6, a completely specified satisfying solution is returned. As a result, the produced

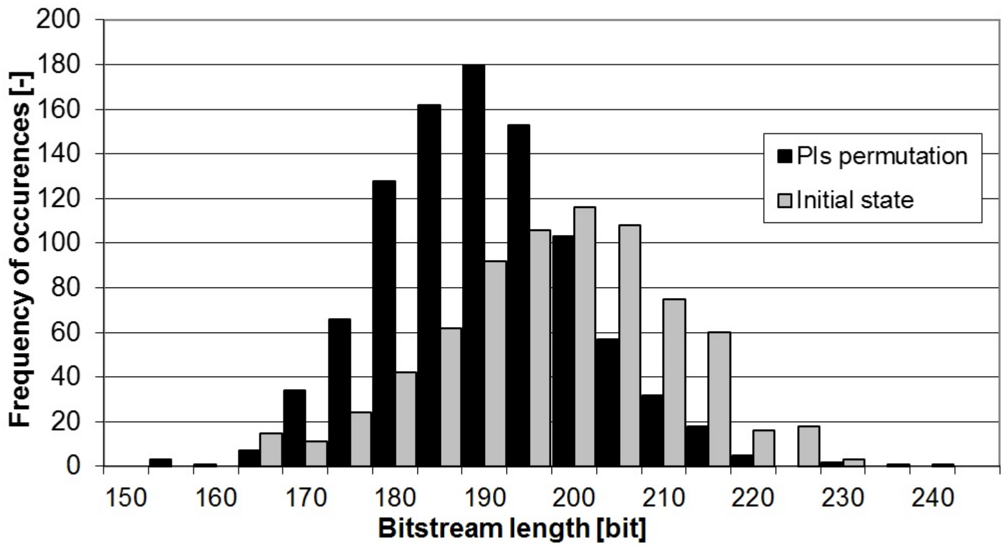

Figure 5.12: The influence of different initial test pattern and PIs permutation on bitstream length distribution for the SAT-Compress algorithm (c1355).

test pattern (which also imposes constraints on the subsequent patterns) is completely specified too. Then don't care values appear only in *"link patterns"* and they are obtained by the pattern shifting (17), thus they occur at the *"tail"* of the constraints cube only.

However, a less specified SAT solution can be obtained, yielding fewer constraints. Possibilities of doing so will be discussed in this section.

5.3.1 Special SAT-Solvers

There is a group of special optimization SAT solvers that minimize the number of assigned variables in the solution. So called *"minimal models"* [132], [133], [134] or *"prime implicants"* [103] are computed here. There are also techniques converting the optimization SAT problem to Integer Linear Programming (ILP) [100].

These SAT solvers return a satisfying assignment of variables as a solution, while the total number of variables assigned a value is minimized.

The SAT instances used in SAT-Compress (and all SAT-based ATPGs) contain variables representing the circuit primary inputs (n in Fig. 5.6), but also incomparably many more other variables describing the circuit structure, fault propagation, etc. [1], [2]. However, the constraints are derived from values of primary inputs only (line 10 in Fig. 5.6); values of the other variables are of no interest in the overall algorithm. Therefore, minimizing the number of assigned primary input variables only makes sense. All the above mentioned techniques minimize the number of *all* assigned variables. This renders those techniques almost useless for our purpose, since there is no guarantee (or at least a promise) that the number of assigned *input* variables, which form a very small subset, will be minimal. For this reason, these techniques will not be studied in this thesis, and more suitable techniques will be devised instead.

5.3.2 Pseudo-Boolean Optimization

Similarly to [100], the optimization SAT problem can be converted to Pseudo-Boolean optimization (PBO) and solved by available efficient PBO solvers, like MiniSAT+ [156]. The principles of the conversion will be described in this subsection.

For our purposes, it is convenient to obtain a maximally unspecified test vector that is a solution with *maximum of unspecified values* at primary inputs. This criterion is often phrased as *maximum don't cares.* As we show here, such a use collides with don't cares (DC) in function description. To avoid confusion, we use the term *unspecified value* in this section, and use the symbol U for such value.

For better understanding, we will start with a conversion of a MIN-SAT problem, where the number of *variables assigned* to "1" is minimized, to PBO:

1. Let $x_1, \dots, x_m$ be variables of the original MIN-SAT problem.
2. For each clause $(l_1 + l_2 + \dots + l_j)$, where l_i are individual literals (variables or their negations) construct an inequality $l_1 + l_2 + \dots + l_j \geq 1$.
3. If a literal $l_i = x_k$ (variable in its direct form), substitute $l_i = x_k$ in the inequality.
4. If a literal $l_i = \overline{x_k}$ (variable in its negated form), substitute $l_i = (1 - x_k)$.
5. Form the optimization criterion as $x_1 + x_2 + \dots + x_m = min$.

Example:

Let us have a CNF formula of 3 variables:

$$(x_1 + x_2)(x_2 + x_3)(\overline{x_2} + \overline{x_3}) \tag{5.1}$$

It will be transformed into the following PBO formulation:

$$\begin{aligned} x_1 + x_2 &\geq 1 \\ x_2 + x_3 &\geq 1 \\ (1 - x_2) + (1 - x_2) \geq 1 \Rightarrow -x_2 - x_3 &\geq -1 \\ x_1 + x_2 + x_3 &= min \end{aligned} \tag{5.2}$$

There are three satisfying solutions of the SAT instance:

$$\begin{aligned} x_1 = 0, x_2 = 1, x_3 = 0 \\ x_1 = 1, x_2 = 1, x_3 = 0 \\ x_1 = 1, x_2 = 0, x_3 = 1 \end{aligned} \tag{5.3}$$

When solved as PBO, a single solution minimizing the number of variables assigned to "1" will be returned:

$$x_1 = 0, x_2 = 1, x_3 = 0 \tag{5.4}$$

Still, all the variables are in the Boolean domain, while we need to encode unspecified values. For this purpose, we must use two Boolean variables to encode each literal, for example, as shown in Tab. 5.3.

Table 5.3: Literals transcription

x_i	x_i^V	x_i^A
0	0	1
1	1	1
unassigned	*any*	0

In this encoding, x_i^A value indicates whether the particular variable is assigned a value, x_i^V is then the value.

The optimization criterion can be then formed as:

$$x_1^A + x_2^A + \cdots + x_{n-1}^A = min \tag{5.5}$$

where $x_1, \ldots, x_{n-1}$ are primary inputs.

Detecting a fault means to control *specified* values in the circuit, and to observe *specified* values at outputs. Hence, the propagation of unspecified values must be observed, and *every* original CNF variable must be doubled in PBO.

The CNF is then rewritten into a PBO instance in a straightforward way, as shown in the Example above. The solution of the PBO instance maximizes the number of unspecified PI values, i.e., a maximum of test don't cares is obtained.

During the circuit-to-CNF conversion, characteristic functions of all gates in the circuit in the CNF form are added to the SAT instance. For a gate with inputs $x_1, \ldots, x_m$ and output y, the signature of the characteristic function F is

$$F : \{0,1\}^{m+1} \rightarrow \{0,1\} \tag{5.6}$$

For our problem, we need to take undefined value U into account, and the function F becomes

$$F : \{0,1,U\}^{m+1} \rightarrow \{0,1\} \tag{5.7}$$

The strategy is to calculate F in some form, then to encode it by Tab. 5.3 into

$$F : \{0,1\}^{2m+2} \rightarrow \{0,1\} \tag{5.8}$$

or, alternatively, into two functions

$$\begin{aligned} F^V &: \{0,1\}^{2m+1} \rightarrow \{0,1\} \\ F^A &: \{0,1\}^{2m+1} \rightarrow \{0,1\} \end{aligned} \tag{5.9}$$

which have y^V resp. y^A as the last argument, and to convert them to CNF form.

The main task is to find a concise and complete representation of F. By completeness we mean that all possible combinations at input and output are covered, so that the origin, propagation, and termination of undefined values can be calculated. For this purpose, we adapted *D-intersection* [12]. Because we represent F as a set of terms in a tabular form, Tab. 5.4 includes also the '-' symbol. Notice that incompatibility cannot occur here.

Table 5.4: Symbol intersection

	0	1	-	U
0	0	U	0	U
1	U	1	1	U
-	0	1	-	U
U	U	U	U	U

The complete algorithm for generation of a CNF for a given gate is shown in Fig. 5.13. Let us assume that *gate* is a table describing the on-set of a completely specified Boolean function with one output, and that the columns of the table are labeled $x_1, \ldots, x_{m-1}, y$. Furthermore, if t is a term, let *t[j]* be the symbol of t in the column labelled *j*.

The algorithm has four main phases. The first one (lines 1 to 5) derives the characteristic function. Lines 6 to 16 are the main phase, which adds terms describing the behavior of undefined values to the function. Finally, the third phase (line 18) encodes the table and phase four (19-27) outputs the resulting CNF, using CNF and DNF duality. Before phase four, F can be split into F^V and F^A. In many cases, the resulting functions are smaller and easier to complement.

For simplification of the truth table and, more importantly, off-set computation (complementation), the Espresso minimizer is used [8].

The algorithm is as feasible as Espresso minimization [8] and off-set generation are. Large XOR gates are difficult and have large characteristic functions, but are manageable to 10 inputs. Specialized algorithms can be devised for even larger gates, this one, however, has the advantage of being universal.

A complete example of a 2-NAND gate to CNF conversion is shown in Fig. 5.14.

For purposes of the test compression algorithm, a library of CNF descriptions for every supported gate is created using the procedure from Fig. 5.13. Thus, the conversion is run only once.

```
CNFlib(gate)
// generate the characteristic function
minimize gate                              // espresso
add the off-set to gate  // espresso
move the output column to input columns of gate
put all 1s into the output column of gate
// calculate intersections
do {
   for each unordered pair ($t_1$, $t_2$) of terms from gate {
     let s be the intersection of $t_1$[y] and $t_2$[y]
     if (s == U) {
        let t be the intersection of $t_1$ and $t_2$
        if t is not in gate
           insert t into gate with output symbol 1
     }
   }
} while new terms are added to gate
// encode and convert to Boolean domain
encode gate using 0giving F
// produce CNF
turn F into off-set description            // espresso
for each term t in F {
   start a new clause
   for each column label j {
     if (t[j] == 0) output j
     if (t[j] == 1) output ¬j
   }
}
```

Figure 5.13: An algorithm generating the CNF characteristic function of a library gate for gate-to-CNF transformation with encoded undefined values.

5.3.3 Injection Techniques

Don't cares can also be obtained by *"injection"* into a completely specified vector (completely specified SAT solution). The most straightforward injection method will be to try to inject don't cares (*"unassign"* variable values) into a completely specified SAT solution, while checking whether the incompletely specified solution is still a satisfying one, under all assignments of don't cares. However, this would require the number of SAT-solver calls exponential with the number of injected don't cares. This makes this approach impractical.

In SAT-Compress, we can benefit from the nature of the problem. Indeed, we primarily require a cube that covers a particular fault as a solution. Hence, the above exponential-time satisfiability checking job can be accomplished by symbolic fault simulation [157], which can be conducted in polynomial time. This idea can be extended further more - we even need not insist on covering the fault the CNF was constructed for; a test cube covering *any* other (not yet covered) fault is a no less valuable solution.

x_1	x_2	y
0	0	1
1	0	1
0	1	1

a)

x_1	x_2	y	
-	0	1	1
0	-	1	1
1	1	0	1

b)

x_1	x_2	y	
-	0	1	1
0	-	1	1
1	1	0	1
U	1	U	1
1	U	U	1
U	U	U	1

c)

$x_1{}^V$	$x_1{}^A$	$x_2{}^V$	$x_2{}^A$	y^A	
-	0	0	1	1	1
0	1	-	0	1	1
1	1	1	1	1	1
-	0	1	1	0	1
1	1	-	0	0	1
-	0	-	0	0	1

d)

$$\left(x_1^A + x_2^A + \overline{y^A}\right)\left(x_2^V + \overline{x_2^A} + y_A\right)\left(x_1^A + \overline{x_2^V} + \overline{y^A}\right)$$
$$\left(x_1^V + \overline{x_1^A} + \overline{x_2^A}\right)\left(x_1^V + \overline{x_1^A} + y^A\right)\left(\overline{x_1^V} + x_2^A + \overline{y^A}\right)$$
$$\left(\overline{x_1^A} + x_2^V + \overline{x_2^A}\right)\left(\overline{x_1^A} + \overline{x_2^A} + y^A\right)$$

e)

Figure 5.14: Example of a) NAND2 function b) its characteristic function, c) with undefined values propagation, d) encoded F^A, e) F^A in CNF.

Naturally, the more specified the test cube is, the more faults it covers. On the other hand, unspecified bits (don't cares) alleviate the constraints and thus maximize the possible overlap of subsequent patterns. Thus, there are two extreme cases here: either we can try to inject as many don't cares as possible, sacrificing the fault coverage of the pattern (which can drop to covering one fault only), or we can try to inject only don't cares retaining the original coverage. Even in the latter case, some don't cares can be injected. Any compromise between these two extremes can be used, by driving the injection by some factor. This will be denoted as *CL* in this thesis, the *Coverage Loss*.

The don't cares injection algorithm is shown in Fig. 5.15. It is supposed to be run after a completely specified SAT solution in Algorithm from Fig. 5.6 is obtained (step 8). The SAT solution (s), the constraint cube (c), the fault-list (*FL*) and the *CL* parameter are the inputs to this procedure.

First, the number of faults detected by s is determined by symbolic fault simulation (1) [157]. Then the test cube (s) is tried for don't care injection in a greedy way (5). Don't cares can be injected into positions allowed by the constraint cube only (4). If the number of faults covered by the resulting cube does not sink below the *CL* factor, the injection is made permanent (7). The procedure returns a test cube with don't cares injected, while its fault coverage is no more than by the *CL* factor less than the coverage of the original one.

This procedure is greedy and its complexity is polynomial with the circuit size (depending on the fault simulation subroutine used). Therefore, it imposes no big run-time overhead.

Note the two extremes: for *CL = 0*, no fault coverage drop is allowed. This technique will be referred to as *Coverage Preserving Don't Care Injection* (CPDCI) [A.7]. For *CL* approaching 1, maximum don't cares are injected, while the fault coverage may drop to one fault only.

Summarized, low values of *CL* represent cases, where the coverage is not lost by the pattern, however less don't cares are injected. High *CL* values induce injecting more don't cares, at expense of losing fault coverage of the pattern. The issue of *CL* choice will be further discussed in the following section.

Note that the *CL* values in the experimental section will be represented in percents (0-100%).

```
InjectDCs(s, c, FL, CL)
1   $d_1$ = |detected_by_simulation(c)|
2   s_tmp = s
3   for (i = 0; i < n; i++) {
4       if ( c[i] == DC ) {
5           s_tmp[i] = DC
6           $d_2$ = |detected_by_simulation(s_tmp)|
7           if ( $(d_1 - d_2) / d_1 \le CL$ ) s = s_tmp
8       }
9   }
10  return s
```

Figure 5.15: The don't cares injection algorithm.

5.3.4 Experimental Evaluation of Don't Care Processing

5.3.4.1 Choice of the CL parameter in the simulation-based don't care injection

It is difficult to say intuitively, what *CL* values will produce best results. Low values preserve the fault coverage of test patterns, which may theoretically speed up the whole compression process. Since patterns covering more faults are generated, less patterns (and therefore SAT instances) will be needed to achieve complete fault coverage, and thus the main loop will be shorter. However, these patterns will be rather constrained (low number of don't cares), and thus the chances of a successful overlap decrease.

Conversely, high *CL* values induce many don't cares, the patterns will more likely overlap, however, more patterns would be probably needed to achieve the complete fault coverage.

While the influence of *CL* on the number of generated SAT instances and injected don't cares is quite clear, it is discussable what effects will these two aspects have on the final bitstream length and the compression run-time. Therefore, we have evaluated the influence of the *CL* value on the algorithm execution experimentally.

Table 5.5: Influence of losing fault coverage

CL [%]	*DCs injected*	*SAT*	*Bits*	*Time [s]*
0	3410.7	772.0	822.0	269.3
5	3436.7	771.8	821.8	262.1
10	3555.6	775.5	825.5	261.0
15	3563.1	773.5	823.5	247.2
20	3774.9	798.2	848.2	252.0
25	3913.4	810.0	860.0	251.6
30	3985.1	815.4	865.4	252.1
35	4303.1	848.6	898.6	260.9
40	4406.7	848.7	898.7	261.8
45	4512.9	857.9	907.9	261.6
50	5145.0	937.4	987.4	255.3
55	5210.8	943.4	993.4	254.6
60	5497.9	970.5	1020.5	267.9
65	5604.0	977.7	1027.7	266.7
70	6061.7	1028.8	1078.8	275.1
75	6493.1	1079.3	1129.3	291.0
80	6701.7	1101.3	1151.3	282.0
85	7108.0	1132.6	1182.6	302.8
90	7768.5	1187.1	1237.1	323.7
95	8598.3	1241.0	1291.0	338.3
$\rightarrow$ 100	9743.6	1297.6	1347.6	364.5

The results for one representative ISCAS'85 [87] benchmark circuit c3540 are shown in Tab. 5.5. Here the SAT Compress algorithm was run with different values of the *CL* parameter and the absolute numbers of injected don't cares (*"DC injected"*), the absolute numbers of SAT instances solved (*"SATs"*), the final bitstream length (*"Bits"*), and the compression run-time (*"Time [s]"*) were measured. The values were obtained from averaging values of 30 runs with random initial patterns, to diminish the influence of randomness and obtain more precise results (see previous chapter).

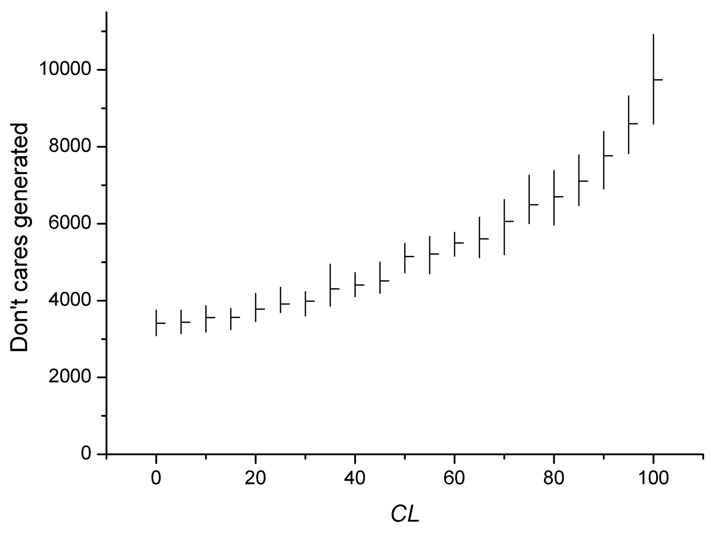

Figure 5.16: Influence of *CL* on the total number of injected don't cares.

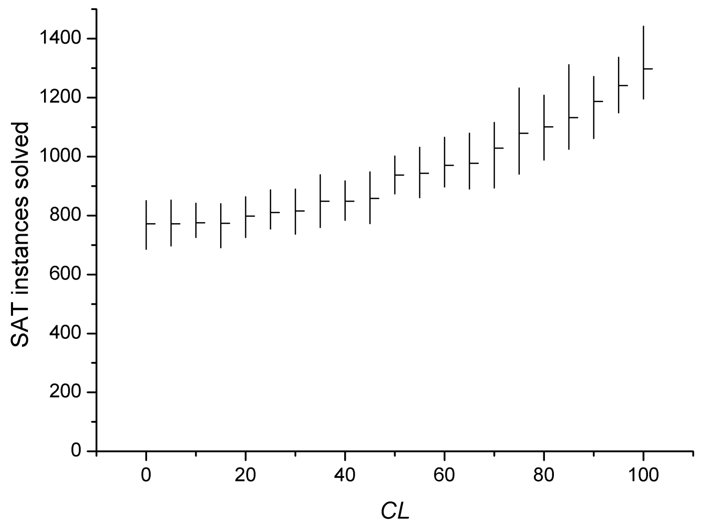

Figure 5.17: Influence of *CL* on the total number of SAT instances solved.

These results are also visualized in Fig. 5.16 - Fig. 5.19, more precisely. Here the minimum, maximum, and average values from the 30 runs are shown.

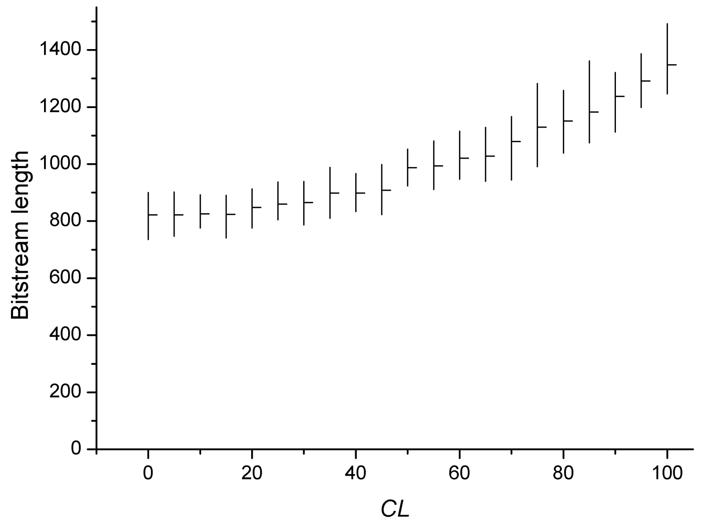

Figure 5.18: Influence of *CL* on the generated bitstream length.

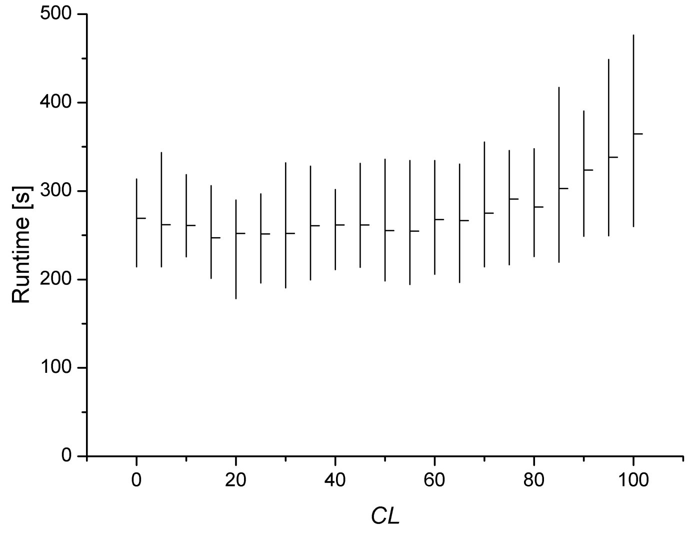

Figure 5.19: Influence of *CL* on the compression run-time.

We can see that the initial assumptions were confirmed: the number of injected don't cares monotonously grows with increasing *CL*, while the number of solved SAT instances increases too.

The most important observation concerns the final bitstream length: the average bitstream length *monotonously increases* with *CL* (see Fig. 5.18), with best results obtained for *CL* = *0*, i.e., the CPDCI technique. Also the run-time decreases for low values of *CL* (Fig. 5.19).

Similar experiments were performed on many other benchmark circuits and exactly the same behavior was observed in all cases. Results for some other ISCAS benchmarks are shown in Tab. 5.6. The final bitstream length only was measured. The data was obtained by averaging 30 runs with random initial patterns, the values are rounded to integer values.

This experiment has shown that no fault coverage of every single pattern should be sacrificed, even though more don't cares would be injected otherwise. Therefore, the usage of the CPDCI technique from [A.7] is fully justified; there is no need for looking for a compromise between the number of injected don't cares and the fault coverage.

Note that the extreme two cases, *CL* = *0* and *CL* → *100%* represent the completely informed and completely uninformed don't care injection techniques, respectively.

The comparison of the original SAT-Compress algorithm and the SAT-Compress algorithm with CPDCI for two illustrative benchmark circuits (c432 and c880) is shown in Fig. 5.20 and Fig. 5.21. Algorithms were executed 5,000-times, each time with a randomly generated initial pattern. The frequencies of occurrence of the resulting bitstreams of different lengths (the x-axis) are shown, both for the basic SAT-Compress (Fig. 5.6) and the SAT Compress augmented with CPDCI.

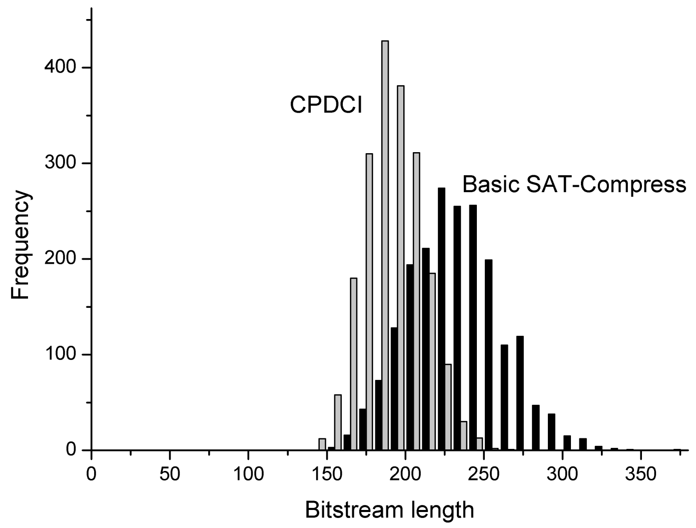

Figure 5.20: Frequency of bitstream lengths distribution (c432).

Table 5.6: Influence of losing fault coverage - final bitstream lengths for more circuits

Circuit / CL [%]	0	10	20	30	40	50	60	70	80	90	100
br1	503	509	518	550	575	623	649	685	716	746	754
c1908	545	538	540	542	551	551	551	560	563	577	609
c2670	1778	1800	1850	1869	1891	1927	1948	1927	1945	2021	2030
c3540	822	826	848	865	899	987	1021	1079	1151	1237	1348
c5315	773	772	774	764	760	783	795	823	866	921	1179
c7552	3608	3620	3630	3700	3706	3948	3934	3971	4080	4195	4398
chkn	3611	3636	3677	3798	3964	4526	4604	4921	5310	5558	5617
comp	559	555	563	562	585	613	615	624	655	650	655
duke2	1010	1018	1043	1072	1080	1146	1177	1201	1262	1310	1349
example2	630	622	633	643	647	685	691	709	728	758	778
frg1	2156	2171	2186	2202	2246	2349	2379	2459	2544	2584	2635
i2	5179	5173	5157	5185	5225	5723	5730	5735	5795	5747	5747
i4	898	899	904	908	911	918	930	949	959	950	962
in0	790	814	840	863	900	931	958	965	980	1008	1025
in2	731	733	730	735	742	776	764	787	822	875	923
in4	865	863	868	899	917	957	990	1044	1085	1145	1202
in5	701	709	731	751	816	930	978	1048	1135	1206	1235
in6	993	1004	1003	1017	1022	1060	1062	1079	1111	1137	1151
jbp	840	856	849	870	874	913	928	947	993	1033	1080
s1196	862	864	866	878	900	932	935	949	961	976	1011
s1238	886	886	887	902	925	942	935	963	982	1002	1000
s1423	608	626	635	645	684	719	718	746	781	804	814
s420.1	600	595	605	640	627	663	675	707	714	732	741
s5378	1920	1933	1954	1982	2030	2140	2151	2203	2275	2301	2344
s820	669	668	669	673	672	705	712	726	747	783	821
s832	665	651	661	673	660	687	694	722	746	771	807
s838	986	978	1022	1003	1036	1190	1238	1262	1302	1358	1337
s838.1	1620	1638	1668	1655	1657	1810	1848	1800	2002	1979	2022

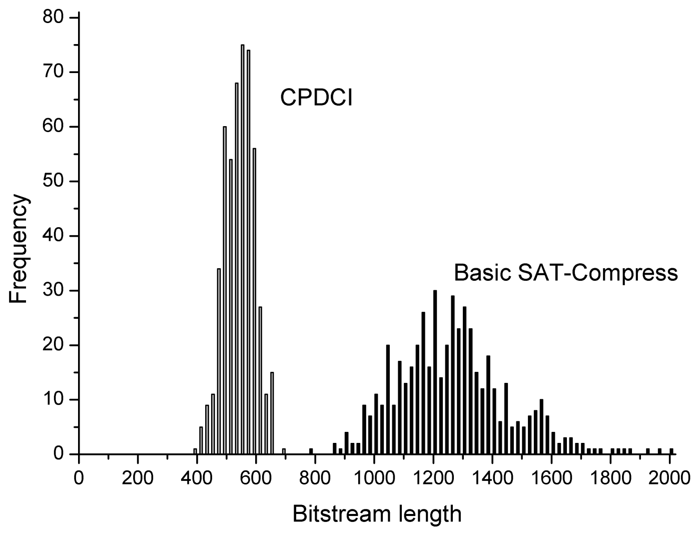

Figure 5.21: Frequency of bitstream lengths distribution (c880).

We can see that the histograms follow the Gaussian distribution, which is expectable. More importantly, the two distributions have different mean values, advantageously to the CPDCI. CPDCI also has smaller standard deviation, thus randomness plays a smaller role here, making CPDCI more robust than the standard SAT-Compress [155]. Nevertheless, the influence of randomness is still crucial (even though reduced in the CPDCI case), and worse results can be obtained by CPDCI accidentally, see Fig. 5.20, where the histograms overlap. Similar frequency distributions were observed for different orderings of faults and don't care injections.

5.3.4.2 Comparison of CPDCI and PBO

As a consequence of the previous experiments, uninformed SAT solvers producing don't care values, including the PBO based technique from subsection 5.3.2 seem to be compromised. Don't cares must be injected with care (in our case, the fault coverage is of the major concern), and definitely not only their number in the SAT solution must be the optimization criterion.

To confirm this, we have made a comparison with the PBO-based technique. The PBO technique produces test patterns with the real maximum of don't cares, however in an uninformed way, too. The results for some benchmark circuits [89], [158] and the three processes (uninformed don't care injection, CPDCI, and PBO) are shown in Tab. 5.7. The final bitstream lengths and the average percentages of test don't cares (out of the number of PIs) are shown. The values were obtained from 1,000 randomized measurements (random

Table 5.7: Comparison of CPDCI and PBO

Circuit	*SAT-Compress, CL → 100%*		*SAT-Compress, CL → = 0 (CPDCI)*		*PBO*	
	Bits	*DCs [%]*	*Bits*	*DCs [%]*	*Bits*	*DCs [%]*
5xp1	155.58	8.56	102.32	0.52	106.82	1.82
b03_C	139.29	12.7	113.12	2.31	111.71	5.80
b06_C	39.78	30.75	35.27	27.23	37.45	39.61
b9	271.23	9.11	215.10	4.14	217.76	9.94
c1355	292.18	1.80	265.57	0.29	264.09	0.79
c432	245.26	9.50	185.93	5.11	193.58	13.70
c499	222.66	2.18	198.28	0.29	200.74	0.90
c8	363.39	24.05	276.09	15.00	281.74	36.32
dc2	133.97	14.44	91.24	8.70	98.20	16.65
f51m	273.02	12.44	163.93	3.35	167.55	5.93
cht	156.09	6.76	133.98	4.94	138.03	6.77
i1	189.25	8.74	152.75	6.10	164.25	21.36
i3	392.22	8.55	349.81	4.83	361.34	13.15
average	221.07	11.51	175.65	6.37	180.25	13.29

initial pattern) and averaged (see subsubsection 5.3.4.1). The average values over all the circuits are computed in the last table row.

We can see that even though maximum of don't cares is obtained by PBO, the resulting bitstreams are typically longer than those obtained by CPDCI.

5.3.4.3 Comparison of SAT-Compress variants

In this subsection we present a more thorough comparison of the original SAT-Compress algorithm (Fig. 5.6) and SAT Compress extended by don't care injection. The experiments were conducted using a subset of the ISCAS benchmark circuits [87], [88].

The results are shown in Tab. 5.8. After the circuit name, the number of its faults is given, as the measure of the circuit complexity. Then results of three variants of SAT-Compress are shown: the original SAT-Compress, i.e., without don't care injection, the uninformed don't care injection (*CL* $\rightarrow$ *100%*), and the informed one (*CL* = *0*, CPDCI). The absolute numbers of injected don't cares (where applicable), the lengths of the final bitstreams, relative bitstream length improvements w.r.t. the original SAT Compress, and the compression times are shown. Average values are shown in the last table row.

We can see that the don't care injection helps to reduce both the final bitstream length and the test compression time. Even the uninformed don't care injection significantly reduces both in some cases. On the other hand, in some cases the injected don't cares make the solution worse. This indicates that the uninformed injection can sometimes do more bad than good. The average bitstream size reduction is positive, by 15%.

Finally, the CPDCI technique reduces the bitstream length by 31% on average and the deteriorating cases are only rare and not significant. This experiment justifies the CPDCI one last time.

5.3.4.4 Summary comparison results

A comparison of the basic SAT-Compress algorithm and its extension by the *Coverage Preserving Don't Care Injection* technique (CPDCI) will be presented in this subsection. The results for some selected circuits are shown in Tab. 5.9.

The first column of the table (*"Circuit"*) represents the name of the benchmark circuit. The second column *"Faults"* gives the number of faults in the circuit, which reflects its size. The next two columns *"Bits"* and *"Time"* represent the number of bits of the compressed bitstream and the time spent by compression by the basic SAT-Compress algorithm. The next columns show results for the SAT-Compress algorithm with the CPDCI technique employed. The lengths of the compressed bitstreams and the compression times are shown there too. The percentage test length and time improvements w.r.t. the basic SAT Compress algorithm are shown in the *"Bits impr."* and *"Time impr."* columns.

Furthermore, the column *"DCs tried"* shows the absolute numbers of care bits tried for DCs injection and the *"DCs inj."* column the number of successfully injected bits. The percentage of successfully injected don't cares is then shown in the *"Success"* column.

Table 5.8: SAT-compress variants

Circuit	*Faults*	*No don't cares*		*CL → 100%*				*CL = 0 (CPDCI)*			
		Bits	*Time [s]*	*DCs*	*Bits*	*Impr. [%]*	*Time [s]*	*DCs*	*Bits*	*Impr. [%]*	*Time [s]*
c1355	1566	330	29.3	159	367	-11	31.3	19	334	-1	28.8
c1908	1869	660	121.6	371	583	12	106.2	92	538	18	110.5
c2670	2629	3758	1714.1	18934	1977	47	738.3	9905	1874	50	1050.0
c3540	3291	3188	3600.4	9711	1426	55	1075.2	3358	774	76	834.6
c432	520	209	3.0	460	185	11	2.7	234	176	16	2.9
c499	750	182	3.1	154	241	-32	3.8	28	219	-20	3.5
c5315	5291	1194	698.0	19137	1207	-1	600.3	1750	795	33	724.5
c7552	7419	6408	7856.4	21741	4314	33	4149.1	6030	3819	40	5878.1
c880	942	1134	107.1	3747	702	38	45.2	1465	458	60	25.9
s1196	1242	2430	262.1	5993	1066	56	117.0	3553	864	64	81.2
s1488	1486	502	61.0	170	662	-32	96.2	11	557	-11	63.3
s208	215	185	0.9	329	207	-12	0.9	129	195	-5	0.7
s298	308	243	1.2	613	166	32	1.0	337	140	42	0.8
s349	348	127	1.1	620	155	-22	1.2	215	108	15	1.1
s400	418	212	1.4	654	205	3	1.2	230	144	32	0.8
s510	564	180	1.7	126	232	-29	2.2	2	165	8	1.6
s641	463	1186	27.5	4223	613	48	20.2	1972	493	58	13.9
s713	543	1456	35.3	4440	598	59	18.3	1894	464	68	14.3
s820	850	700	25.7	142	855	-22	38.8	65	664	5	23.4
s953	1079	3360	322.9	5812	959	71	78.6	3693	771	77	48.4
average	1590	1382	743.7	4877	836	15	356.4	1749	678	31	445.4

Finally, results of the test compression tool COMPAS [6], [148], [149] are shown, because it represents the current state-of-the-art and it is based on the same principles (the RESPIN architecture).

The compressed bitstream lengths are given in the *"Bits"* column, the bitstream length differences w.r.t. the proposed CPDCI technique is shown in the last column. COMPAS runtimes are not present, since the experiments were conducted on different platforms, thus they are hardly comparable.

The last row of the table shows average values obtained from all measured 170 benchmark circuits.

We can see that the CPDCI technique can significantly decrease the length of the compressed bitstream and accelerate the algorithm. The bitstream length is reduced by 46.31% on average and the compression time is reduced to 35.18% in comparison with the basic SAT-Compress algorithm. Even the success of the CPDCI technique in don't care injection is remarkable; more than 65% of bits tried were successfully assigned a don't care.

The CPDCI technique increased the efficiency of the SAT-Compress algorithm, both in the compressed test length and test generation run-time. However, there are cases where the extended SAT-Compress algorithm produced worse results, e.g. for the c499 circuit. We assume that this is caused by the random noise introduced by the algorithm, as shown in subsubsection 5.3.4.1.

The scalability of the method is naturally given by the circuit size, particularly the number of faults, but also by its efficiency - the more faults are detected in each algorithm step, the faster is the overall algorithm. Therefore, the CPDCI technique maximizing the number of covered faults in each step also significantly reduces the runtime.

In comparison with COMPAS we reach a 6% improvement on average. There are benchmarks, for which SAT-Compress strikingly overcomes COMPASS (e.g., c1355, c2670). For some benchmarks COMPAS wins, however, the differences are not so large (except of some extreme cases, like s35932). This is probably due to a huge amount of randomness introduced into the ATPG process, as shown in subsubsection 5.3.4.1. We can conclude that these two techniques are competitive.

5.4 Processing of SAT instances in Serial Compression

The SAT-Compress algorithm deals with repeated processing of the same SAT instances in CNF with different constraints. The processed CNFs are often unsatisfiable with given constraints. As they are repeatedly generated and solved with different constraints, the CTPG process becomes time-consuming [A.10], [144]. Thus, the CTPG process itself and possible techniques of its acceleration have been analyzed. We discuss the CNFs manipulation and their filtering based on satisfiability, analyze the efficiency of the proposed techniques and evaluate their usability in SAT-based CTPG algorithms.

All measurements were performed on a CPU Intel Core 2 Duo - 1,8GHz with 1GB RAM. MiniSat v1.14 [129] has been used as the SAT solver. Experiments have been performed

Table 5.9: Experimental results for the basic SAT-Compress algorithm and CPDCI

Circ. name	Faults	SAT-Compress		SAT-Compress with CPDCI							COMPAS	
		Bits [-]	Time [s]	Bits [-]	Bits Impr. [%]	Time [s]	Time impr. [%]	DCs tried [-]	DCs inj. [-]	Success [%]	Bits [-]	Diff. [%]
alu4	6435	3349	1994.97	3048	8.99	1773.53	11.10	3380	349	10.33	-	-
b04_C	1666	5408	463.67	910	83.17	88.25	80.97	7659	6876	89.78	-	-
b05_C	1928	1091	90.95	631	42.16	55.57	38.90	1593	1018	63.90	-	-
b07_C	1084	997	9.89	706	29.19	7.88	20.32	1444	895	61.98	-	-
b11_C	1675	863	41.90	562	34.88	32.00	23.63	1557	1084	69.62	-	-
c1355	1566	330	13.40	334	-1.21	13.43	-0.22	312	19	6.09	1040	67.88
c1908	1869	607	44.66	495	18.45	36.71	17.80	542	82	15.13	1009	50.94
c2670	2629	3103	556.27	1806	41.80	387.30	30.38	11276	9791	86.83	6553	72.44
c3540	3291	3422	1618.65	833	75.66	323.90	79.99	4146	3415	82.37	747	-11.51
c432	520	209	1.39	156	25.36	1.28	7.91	368	256	69.5	195	20.00
c499	750	182	1.51	219	-20.33	1.67	-10.60	206	28	13.59	260	15.77
c5315	5291	1205	261.30	815	32.37	275.89	-5.58	2410	1812	75.19	1255	35.06
c7552	7419	6581	2739.73	3522	46.48	1902.73	30.55	9029	5998	66.43	6005	41.35
c880	942	1195	35.71	614	48.62	15.16	57.55	2250	1765	78.44	540	-13.70
duke2	1302	1486	56.80	986	33.65	35.13	38.15	1717	810	47.18	-	-
ex5p	5430	276	38.72	276	0.00	42.12	-8.78	268	0	0.00	-	-
intb	1893	2070	220.27	1653	20.14	171.01	22.36	2103	471	22.40	-	-
jbp	1132	2174	41.42	843	61.22	16.69	59.71	2281	1563	68.52	-	-
misex3	9251	3556	5240.01	3467	2.50	5220.65	0.37	3551	100	2.82	-	-
s1196	1242	2487	109.33	876	64.78	36.36	66.74	4292	3474	80.94	740	-18.38
s1238	1286	2705	141.46	876	67.62	40.06	71.68	4926	4105	83.33	741	-18.22
s13207	9664	114390	285075.00	5498	95.19	22678.30	92.04	206673	202598	98.03	4163	-32.07
s1423	1501	1179	46.38	628	46.73	39.53	14.77	2346	1871	79.75	596	-5.37
s15850	11336	77582	147342.00	5734	92.61	22686.30	84.60	179148	174836	97.59	8234	30.36
s344	342	161	0.59	95	40.99	0.47	20.34	280	210	75.00	85	-11.76
s35932	35110	3686	308382.00	4998	-35.59	390677.0	-26.69	2971215	2969101	99.93	1860	-168.71
s382	399	255	0.61	131	48.63	0.39	36.07	258	161	62.40	123	-6.50
s420	430	526	2.81	370	29.66	1.62	42.35	748	463	61.90	352	-5.11
s526n	553	830	5.27	471	43.25	2.70	48.77	1197	785	65.58	344	-36.92
s5378	4511	19847	6765.48	1989	89.98	870.94	87.13	31022	29444	94.91	2148	7.40
s641	463	1335	13.35	469	64.87	5.62	57.90	2282	1919	84.09	397	-18.14
s713	543	1223	11.64	454	62.88	6.08	47.77	2199	1859	84.54	428	-6.07
s820	850	702	10.30	664	5.41	9.97	3.20	692	65	9.39	460	-44.35
s838	857	2078	32.20	955	54.04	19.27	40.16	2957	2242	75.82	920	-3.80
s9234	6475	24395	25844.19	5688	76.68	10238.03	60.39	53308	48599	91.17	11594	50.94
s953	1079	3131	95.99	771	75.38	20.23	78.92	4317	3693	85.55	723	-6.64
t481	2853	5541	1808.29	5147	7.11	1561.09	13.67	5433	304	5.60	-	-
table3	2487	2025	382.11	2085	-2.96	413.89	-8.32	2134	69	3.23	-	-
table5	2384	3191	703.46	2821	11.60	609.92	13.30	3301	547	16.57	-	-
term1	1314	6221	405.79	1418	77.21	102.96	74.63	5443	4089	75.12	-	-
vda	1970	680	31.95	594	12.65	27.24	14.74	652	103	15.80	-	-
vg2	1122	2507	59.94	1403	44.04	32.53	45.73	2430	1093	44.98	-	-
x1	2504	7583	886.07	2953	61.06	354.54	59.99	7689	5013	65.20	-	-
Avg.	3396	7649	23209.59	1585	46.31	13907.19	35.18	86341	85018	65.54	1981	6.14

on a subset of smaller ISCAS'85 [87], '89 [88] and ITC'99 [89] benchmark circuits, because the memory requirements for CNFs storing were unfeasible for bigger circuits.

5.4.1 On-The-Fly Generation vs. Storing in Memory

Generating SAT instances in CNF takes on average 80% of test generation time in SAT-based ATPGs [1], [2], [47], [73] and can also cause a significant time overhead in the CTPG process. When used repeatedly, SAT instances can be generated on-the-fly, or they can be pre-generated and stored in memory. Either original CNFs are stored, or the CNFs are simplified by solution set preserving reductions [A.11], to reduce memory requirements.

Under common fault models, a fault causes the value of a signal to differ between the faulty and fault-free copy of the circuit in the SAT ATPG process. The values are determined by the fault model. It suffices to inject them into the corresponding CNFs using unit clauses, however, this opens up a way to simplify the CNFs. Specifically, values of other variables common to all solutions can be discovered.

Repeated application of unit clause elimination identifies most of constant variables. The rest of them are detected by fixing the variable to a constant value (both 0 and 1) and solving the SAT problem to evaluate if both instances are satisfiable or unsatisfiable, see [159]. The number of clauses can be reduced by removing duplicities, clause absorption, and by creating resolution terms [A.11]. All these reductions preserve all SAT solutions.

Our experimental results show that on the average 60% of variables and 65% of clauses can be removed by solution-set-preserving reductions [A.11], see chapter 4 for details. Thus it can be possible to store all CNF instances or their subset in memory, decrease the number of repeatedly generated CNF instances, and accelerate the CTPG process.

In the on-the-fly approach, the CNFs are repeatedly generated while they are differently constrained in the test generation process. It is obvious that memory requirements are negligible. On the other hand, such a repeated generation of CNFs can increase the test generation time significantly [A.10].

The next technique is based on storing of all processed CNFs in memory. CNFs for each fault are generated only once in the initial part of the algorithm and stored in memory. The time overhead incurred by repeated CNF generation is reduced. However, constraints change in the test generation process, thus original (unconstrained) CNFs must be repeatedly loaded into the SAT solver.

Loading of the CNFs into the SAT solver should cause much less time overhead, but the number of literals stored in memory can be unfeasible for larger circuits. The number of stored literals can be further reduced by the solution set preserving SAT reductions described in chapter 4 [A.11], [114].

Experimental results and a comparison of the three techniques (generation on-the-fly, processing of the original and reduced CNFs stored in memory) of CNFs processing is presented in Tab. 5.10. The first column of the table *"Circ. name"* represents the name of the benchmark circuit from ISCAS'85 [87] or '89 [88]. Differences between processing of the CNFs on-the-fly and storing of non-reduced/reduced CNFs are shown in the three columns. The *"CNF"* columns indicate the time spent by a CNF generation or loading of stored CNFs from the memory. The *"SAT"* columns represent the time spent by a SAT solving of the processed CNFs. Columns *"SIM"* show the time spent by simulation and columns *"SUM"* indicate the total time consumptions of the algorithm. For methods, where storing of CNFs was employed, the stored CNF literals count (*"Lit. count"*) and the time spent for the CNFs generation eventual reduction and storing is shown (*"Store"*). The last row of the table (*"Avg."*) represents average values of all columns.

First, let us focus on the time spent by CNFs generation. Experimental measurements show that loading of the stored CNF instances is in all cases faster than their generation on-the-fly. In the case of the benchmark circuit c3540, the time of CNFs generation is 2205 seconds, whereas loading of the previously generated CNFs stored in the memory is made in 1112 seconds and for reduced CNFs it takes only 436.8 seconds.

The time consumption of CNFs storing seems to be negligible in comparison with CNFs generation and SAT solving. Similar behavior is observed for storing of the reduced CNFs on majority of processed benchmark circuits. However, in some cases, the CNFs reduction increases the time consumption of CNFs storing, and as a result it takes comparable time with the SAT solving. For example, the time of the reductions and storing of the CNFs for the benchmark circuit s1494 is 26.14 seconds, while solving of these CNFs takes 35.54 seconds.

On the other hand, the number of stored literals grows linearly with the size of the circuit (number of gates). For example, the benchmark circuit c3540 consists of 1648 gates and its fault list has 3428 faults. It means that 3428 CNFs must be stored, which is 31,439,618 literals (after reduction). It is obvious that storing the CNFs is unfeasible for large circuits, because of memory consumption. CNF reduction does not improve the situation, because the reduction of the CNFs size was not as significant as we hoped for [A.11].

The average values confirm the previous observations. The time consumption of the CNFs processing can be dramatically decreased by storing the reduced CNFs in memory. The average total time for the CNFs processing indicates, that processing of the stored CNFs is on average 1.34-times faster than its processing on-the-fly. Thus it seems that storing of the CNFs is better than processing of the CNFs on-the-fly, but the memory consumption of the stored literals can be unfeasible. The storing of the reduced CNFs does not decrease the processing time of the CNFs, because the solution set preserving reductions are time consuming for bigger instances.

It can be concluded that for small circuits it is better to store CNFs or reduced CNFs, but this is unfeasible for large circuits, because of high memory requirements. The SAT solving times indicate that generation of the CNFs on-the-fly can be the best way to choose, because it is not limited by the high memory consumption.

5.4.2 Fault Filtering

Constrained test patterns generation algorithms like SAT-Compress must usually solve a great number of constrained SAT instances repeatedly. As mentioned above, it has been observed, that the majority of these instances are unsatisfiable with given constraints (do

Table 5.10: Experimental results for the processing of CNFs

Circ. name	CNFs on-the-fly				Stored CNFs						Stored reduced CNFs					
	CNF	SAT	SIM	SUM	Lit. count	CNF	SAT	SIM	Store	SUM	Lit. count	CNF	SAT	SIM	Store	SUM
	[s]	[s]	[s]	[s]	[-]	[s]	[s]	[s]	[s]	[s]	[-]	[s]	[s]	[s]	[s]	[s]
c432	1.766	1.7969	0.06	3.625	896428	1.031	1.172	0.063	0.484	2.75	835842	0.5	1.15625	0.063	1.172	2.891
c499	1.047	1.8281	0.03	2.906	165789	0.641	1.922	0.063	0.813	3.438	160876	0.391	1.1875	0.078	1.594	3.25
c880	37.2	87.797	12.5	137.5	112133	27.78	93.88	13.77	0.688	136.1	105340	10.95	75.4063	9.547	1.734	97.64
c1355	9.656	20.5	0.38	30.53	678713	7.516	25.17	0.469	4.734	37.89	659375	4.109	23.9531	0.563	14.59	43.22
c1908	39.72	72.938	13.4	126	974343	25.58	64.23	11.42	8.75	110	938495	18.61	60.1719	11.34	46.67	136.8
c2670	212	972.55	3.98	1189	116312	154.8	1047	4.922	8.703	1215	111512	107.1	1213.89	6.422	59.88	1387
c3540	2205	3176.1	403	5784	331944	1112	2201	260.7	42.53	3615	314396	430.8	1730.28	220.8	1322	3704
c5315	27.19	153.59	17.7	198.5	229878	16.31	169.6	16.67	15.52	218.1	224376	16.77	162.172	20.69	50.47	250.1
s420	3.875	8.8438	0.08	12.8	208359	2.031	9.563	0.063	0.125	11.78	146410	1.063	8.51563	0.063	0.703	10.34
s510	0.453	1.25	0.38	2.078	346745	0.156	1.594	0.156	0.219	2.125	266822	0.063	1.26563	0.359	1.531	3.219
s526	1.172	7.25	0.14	8.563	173319	0.516	7.219	0.203	0.109	8.047	130328	0.313	7.0625	0.266	0.375	8.016
s526n	1.094	6.4531	0.17	7.719	173156	0.531	6.141	0.281	0.125	7.078	130075	0.313	6.17188	0.188	0.375	7.047
s641	6.953	21.203	1.17	29.33	539954	5.109	20.72	1.063	0.297	27.19	480537	2.719	19.8438	1.25	1.766	25.58
s713	6.625	18.969	1.27	26.86	674441	4.375	19.22	1.219	0.375	25.19	605613	2.703	18.2031	1.297	2.328	24.53
s820	8.078	34.234	1.97	44.28	451536	5.266	34.92	1.719	0.266	42.17	362319	2.594	33.4531	1.688	3.672	41.41
s832	9.609	37.609	2.81	50.03	462205	4.656	31.94	1.156	0.281	38.03	369538	2.875	30.3438	1.141	3.781	38.14
s838	46.52	126.58	0.2	173.3	705499	27.08	127.5	0.219	0.422	155.3	503436	12.98	109.047	0.391	6.203	128.6
s953	14.95	62.781	3.95	81.69	103712	9.625	61.14	3.688	0.625	75.08	820193	3.781	58.8125	3.938	11.78	78.31
s1196	93.02	230.5	15.9	339.4	213462	60.91	248.5	13.09	1.234	323.7	188413	25.31	199.391	14.2	16.11	255
s1238	110.5	270.66	16.4	397.5	235033	59.8	236.5	15.42	1.375	313	204925	31.23	223.578	15.28	21.02	291.1
s1423	28.03	79.781	5.25	113.1	263552	16.14	108.8	6.406	1.609	133	253852	7.828	68.0781	5.125	5.703	86.73
s1488	3.344	27.234	2.06	32.64	120944	2.234	27.73	1.656	0.734	32.36	100207	1.016	27.75	1.984	25.58	56.33
s1494	3.906	33.141	2.02	39.06	121715	2.609	34.84	2.094	0.75	40.3	100777	1.453	35.5469	2.016	26.14	65.16
Avg.	124.84	237.11	21.94	383.9	444952	67.22	199.11	15.5	3.94	285.78	420879	29.8	178.92	13.85	70.65	293.24

not produce a test pattern). In SAT-Compress, 98% of generated CNFs are unsatisfiable with given constraints on average [A.11]. Generation and solving of these CNFs can cause a significant time overhead. That is why we focused on filtering of faults which lead to such UNSAT instances, in order to accelerate the constrained test patterns generation.

To detect a fault, a set of signals must be set to required values. For a stuck-at fault, it is a single value of a single signal. Given some constraints as a set of fixed values for the primary inputs of the circuit, we can propagate these constant values to other signals in the circuit. If the signal values required to detect a fault conflict with values implied by the input constants, that fault cannot be excited and hence tested.

The advantage of such filtering technique is that constant propagation is done once for a set of constraints, and used repeatedly for all faults. In the circuit domain, it is equivalent to Boolean Constraint Propagation [160], [130], [131] in the SAT domain.

Two fault filtering algorithms are described and experimentally evaluated in the following subsections. An implication filter using static implications to detect unsatisfiability is used in the first approach, which is then extended by dynamic implications in the second.

5.4.2.1 Static Fault Filtering

Many techniques [1], [83], [161], [162] utilize implications in SAT instances to accelerate searching for the satisfied solution. Our implication filters are not meant to accelerate or solve the SAT problem. Their main aim is fast detection of conflicts produced by application of constraints (variable assignments) on the instance of the SAT problem in CNF. These conflicts indicate that the SAT problem is unsatisfiable with given constraints (variable assignments) and their processing (generation and solving) can be skipped.

The static implication filter is based on the observation that ATPG CNFs consist of 70% of 2-literal clauses and 24% of 3-literal clauses [A.11] on average, see chapter 4 for details. Each 2-literal clause can be substituted by two implication rules, e.g., the clause $c \vee e$ corresponds to implications $\neg c \Rightarrow e$ and $\neg e \Rightarrow c$ (see example in Fig. 5.22). When a constraint is applied, a 2-literal clause may become a unit clause, and then the implications serve as an efficient way for constraint propagation [1], [114], [130], [131].

The static fault filter uses a data structure called implication table to store implications. The key of the table is a signal identification and polarity; the value is a list of signals and their polarities to be set.

The table is constructed once for a given fault-free circuit (hence the terms static filtering and static implication). For each gate in the circuit (Fig. 5.22.a), any pair of signals which would form a 2-literal clause in the description of the gate (Fig. 5.22.b) is selected and the corresponding two implications are entered into the table (Fig. 5.22.c).

When any signal is fixed to a constant value, the implication table is used to fix implied values of other signals (Fig. 5.22.d). This is done repeatedly until there are no more signal values to fix.

Constants propagation in a fault-free circuit cannot produce conflicting values for any signal, because initially only values of primary inputs are fixed.

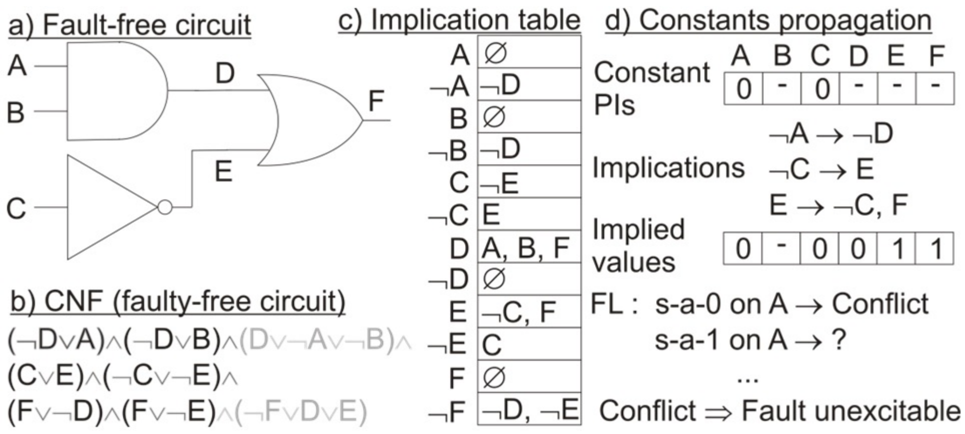

Figure 5.22: Example of the static implication filter.

The implied fixed signal values are used in the way outlined above. In Fig. 5.22, we have constant primary inputs *A=0* and *C=0*. To detect a stuck-at-0 on *D*, we need to set *D=1*. As can be observed in Fig. 5.22.d, this conflicts with the implied value *D=0*, and hence stuck-at-0 on *D* cannot be detected under current constraints and shall be filtered out. On the other hand, stuck-at-1 on *D* is compatible and can be processed further.

The static implication filter is a simple method to check faults for excitability. Its efficiency depends on the number of fixed input signal values and the structure of implications. A high number of applicable implication rules gives us a solid ground for fixing internal signal values and increases our chances to discard a fault before its CNF is generated and proved unsatisfiable. Static fault filtering takes a negligible part of the overall running time, so it can yield in a significant acceleration of the algorithm.

5.4.2.2 Dynamic Fault Filtering

Static filtering uses only implications produced by transformation of 2-literal clauses in the corresponding CNF. Not only a 2-literal clause in a CNF may become a unit clause under given constraints, but, with a sufficient number of constraints, a clause with any number of literals may become. 4-literal clauses are relatively rare, but clauses with 3 literals do occur during SAT ATPG [A.11], [144]. To reflect faults whose unexcitability follows from 3-literal clauses is the task of the dynamic filter.

Unlike the static filter, the dynamic filter cannot rely on a precomputed data structure (hence the term *dynamic*). Instead, it scans the circuit for each set of fixed signal values. Assuming that static filtering has been already done, it searches for triples of signals, which would form a 3-literal clause in the CNF description of a gate. If the values of two signals from the triple are already fixed, it propagates them and fixes the third. This is again done repeatedly until there are no more signal values to fix.

Let us assume constants *A=1*, *B=1* in the circuit in Fig. 5.22, Static implications do not bring any new fixed value (Fig. 5.23.a). The AND gate at the inputs A and B, which would be described by the clause $D \vee \neg A \vee \neg B$, allows the dynamic filter to also set D=1 (Fig. 5.23.b).

It is obvious that further signal values can be fixed. A higher number of fixed values increases the chances to find conflicts and to identify more unexcitable faults than the static filter. However, the time consumption of the dynamic filter is much higher than of the static filter, because the circuit must be searched for new implications for each constant value applied.

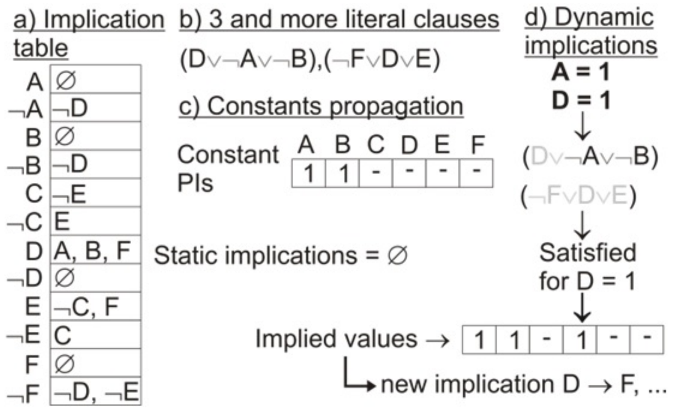

Figure 5.23: Example of the dynamic implication filter.

The pseudocode of the SAT-Compress algorithm equipped with a static and a dynamic filter is outlined in Fig. 5.24. The effort spent in constant propagation is not wasted even in the case of excitable faults; the constants are used as additional constraints to the CNF. Technically, the 2-literal clauses contained in the implication table are not even generated.

5.4.2.3 Experimental Evaluation of Fault Filters

A comparison of filtering techniques is presented in Tab. 5.11. The first column of the table *"Circuit"* represents the name of the benchmark circuit from ISCAS'85 [87], '89 [88] or ITC'99 [89]. Differences between the basic algorithm (SAT-Compress [6]) and its modification with static and dynamic filtering are shown in the three respective columns. The *"Gen."* column represents the total number of generated CNFs and *"Used"* shows the total number of satisfiable CNFs. For algorithms, where static and dynamic filter was employed, the percentage reduction (*"Red."*) of the number of processed CNFs referred to the basic algorithm and the time spent by filtering (*"Filter"*) of the unexcitale faults is shown. The last column *"SUM"* has the same meaning as in Tab. 5.10. The last row of the table *"Avg."* represents an average value of the column.

Experimental results show that fault filtering can accelerate the process of the constrained test patterns generation more than 2-times. For example, the total test patterns compression time of the basic algorithm for the benchmark circuit c3540 is 5,784 seconds,

Table 5.11: Experimental results for the UNSAT filtering

Circuit	Basic algorithm			Basic algorithm + static filter			Basic algorithm + static + dynamic filter		
	Gen.	*Used*	*SUM*	*Red*	*Filter*	*SUM*	*Red*	*Filter*	*SUM*
	[-]	*[-]*	*[s]*	*[%]*	*[s]*	*[s]*	*[%]*	*[s]*	*[s]*
c432	3517	74	3.63	25	0	2.61	64.5	0.3	1.65
c499	3210	73	2.91	9.1	0.02	2.47	49.6	0.94	2.26
c880	70395	181	137.5	46	0.19	82.99	54.6	5.22	79.12
c1355	13236	95	30.54	32	0.02	20.29	70.4	3.22	12.35
c1908	35443	162	126	19	0.13	103.93	47.3	8.81	78.01
c2670	277808	355	1188.98	33	1.45	772.36	52	58.2	624.5
c3540	717888	347	5784	53	4.14	2793.14	62.9	254	2472
c5315	26978	289	198.9	15	0.17	171.17	55.4	88.2	180.2
s298	2782	93	0.93	47	0	0.53	47.8	0.03	0.62
s382	1959	59	1	46	0	0.55	49	0.08	0.61
s400	4088	69	2.06	54	0	0.96	54.9	0.14	1.11
s420	17210	93	12.8	39	0.08	8.08	45	0.58	7.95
s444	2359	59	1.24	58	0	0.53	62.8	0.09	0.6
s510	2899	76	2.08	58	0.02	0.95	59.1	0.16	1.13
s526	15563	134	8.56	49	0.02	4.41	49.8	0.42	5.15
s526n	13901	134	7.71	48	0.06	4.06	49.6	0.38	4.27
s641	20397	136	29.32	37	0.02	17.79	50.2	1.08	16.07
s713	17928	130	26.9	35	0.02	17.29	46.2	1.59	16.31
s820	51351	206	44.25	38	0.13	27.73	38.6	1.09	28.08
s832	56590	203	50.02	39	0.13	30.65	39.5	0.86	30.98
s838	114070	187	173.7	40	0.13	107.69	43.6	7.55	115.57
s953	63076	198	81.75	59	0.06	33.82	67.7	3.73	30.9
s1196	180005	249	339.9	48	0.41	178.41	59.5	18.3	156.3
s1238	213134	247	398.4	49	0.55	206.45	56.3	20.4	192.4
s1423	44892	149	113.05	52	0.17	56.91	59.8	9.41	56.4
s1488	19053	204	32.6	48	0.08	17.4	50.5	4.27	20.6
s1494	22878	201	39.03	51	0.08	19.39	54	4.31	22.79
s5378	378447	502	3216	50	4.25	1649.25	63.3	274	1495.9
s9234	4654444	749	82176.1	42	134	48336	52.6	4014.7	44311.4
s13207	10733919	1109	278511	60	228.4	110828.4	64.1	14130	115520.7
s15850	11160862	980	371963	52	315.1	188644.1	59.5	25019	184213
s35932	2000941	1419	137704	17	402.4	119823.4	20.7	10319	129133.5
s38584	11131264	2256	516020	62	591.2	201443.2	68.3	138870	307178
b03_C	604	48	0.27	42.2	0	0.2	42.5	0.063	0.234
b04_C	265730	198	667	22.6	1.2	519	39.7	25.16	435.5
b05_C	104484	171	319	24.5	0.64	253	41.7	24.39	214.7
b06_C	63	28	0.02	19	0	0.02	22.2	0	0.016
b07_C	36186	116	51.3	49.9	0.13	25.4	65.6	2.813	21.48
b08_C	19145	77	12.4	41.1	0.03	7.5	64.1	0.531	5.172
b09_C	3367	54	2.22	53.8	0.02	1.08	56	0.188	1.172
b10_C	3410	90	1.83	48.6	0.03	1.02	48.9	0.266	1.156
b11_C	56671	143	128	32.8	0.19	87.3	49.9	8.547	64.47
b12_C	157888	337	448	53.1	0.64	222	61.9	34.77	222
b13_C	3661	81	3.53	51.3	0	1.84	53.1	0.422	2.266
Avg.	928778	278.15	30436.12	42	36.65	14707.1	52.4	4200.37	17108.2

```
FL.generate(circuit);                          // FL … fault list
FL.remove_untestable_faults();
TP = T0;                                       // TP … current test pattern
                                               // pick an initial pattern
FL = FL - circuit.fault_simulate (TP);         // remove faults detected by TP
IMTAB.generate (circuit);                      // create implication table
output (TP[0]};                                // put the leftmost bit to output
TP.DC_shift_left();                            // the rightmost TP bit becomes DC
while (!FL.empty()) {                          // loop until all faults detected
    IM = TP;                                   // implied signal values
    IM.static_implications (IMTAB);            // fast static constant propagation
    IM.dynamic_implications (circuit);         // more thorough const propagation
    for each F in FL {                         // find a fault which is detectable
                                               // under current constraints
        if (F.conflict (IM)) {                 // not excitable with current PIs
             continue;                         // skip (filter) unexcitable fault
        }
        CNF.generate (circuit, F);
        CNF.constraint (IM);                   // use propagate consts as constraints
        Y = CNF.solve();
        if (Y.exists()) break;                 // find first testable fault
    }
    if (Y.exists()) {                          // successful search
        TP = Y;
        FL = FL - circuit.fault_simulate (TP);
    }
    output (TP[0]};
    TP.DC_shift_left();
}
output (TP);                                   // output all remaining bits
```

Figure 5.24: The SAT-Compress algorithm with filtering.

while with the static filer it takes 2,793 seconds and with the dynamic filter the total test patterns compression time decreased to 2,472 seconds. The static filter, as a simple fast technique of detecting of the unsatisfiable instances, is highly effective and saves 43% of all the processed unsatisfiable instances on average. Moreover, the dynamic filter is able to detect and save additional 10% of the unsatisfiable CNF instances on average, but it is much more time-consuming than the static filter.

More detailed comparison of filtering techniques for five benchmark circuits is presented in Tab. 5.12. The rows in the Tab. 5.12 have the same meaning as the corresponding columns in Tab. 5.10 and Tab. 5.11. This table shows detailed distribution of the CTPG time over all steps of the SAT-Compress algorithm. The results for presented circuits except of s35932 show that the filters can decrease the time of CNFs generation and their solving in half.

Next, properties of both static and dynamic fault filters have been analyzed for the SAT-Compress algorithm. First, we have measured the number of constraints (fixed PI values) set during the CTPG process. Fig. 5.25 shows an example of constant PIs for the ISCAS'89 benchmark circuit s13207. This circuit has 700 PIs and our measurement shows that on average 497 of them are fixed during the compression. Fig. 5.26 shows the

Table 5.12: Detailed results for the UNSAT filtering

Circuit name		*c3540*	*s13207*	*s15850*	*s35932*	*s38584*
Basic algorithm	*Gen. [-]*	717888	10733919	11160862	2000941	11131264
	Used. [-]	347	1109	980	1419	2256
	CNF [s]	2205	25209	36446	5529.6	19658
	SAT [s]	3176	228267	304751	131884	478943
	SIM [s]	403	25035	30766	290.4	17419
	SUM [s]	5784	278511	371963	137704	516020
Basic algorithm + static	*Red. [%]*	53	60	52	17	62
	Filter [s]	4.41	228.4	315.1	402.4	591.2
	CNF [s]	1041	9869	17406	5154	7392
	SAT [s]	1352	76220	140654	113974	175764
	SIM [s]	396	24511	30269	293	17696
	SUM [s]	2793.41	110828.4	188644.1	119823.4	201443.2
Basic algorithm + static + dynamic	*Red. [%]*	62.9	64.1	59.5	20.7	68.3
	Filter [s]	254	14130	25019	10319	138870
	CNF [s]	829	9048.7	14212	4935	6265
	SAT [s]	993	67857	114844	113585	144431
	SIM [s]	396	24485	30138	294.5	17612
	SUM [s]	2472	115520.7	184213	129133.5	307178

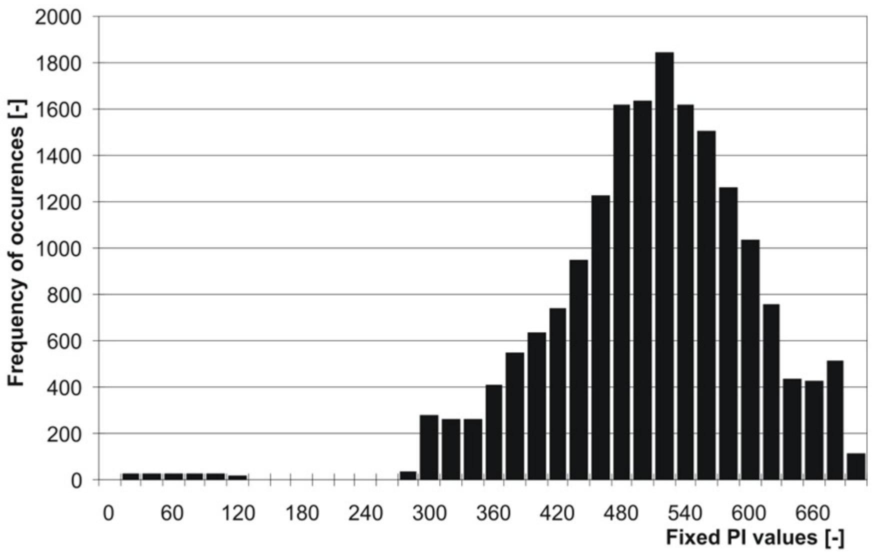

Figure 5.25: Frequency of fixed PI values during CTPG (s13207).

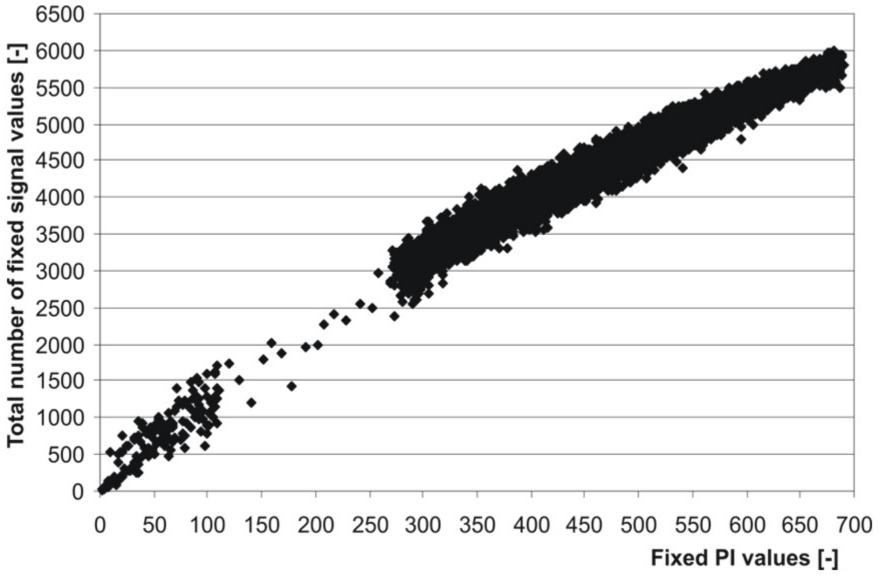

Figure 5.26: The number of values fixed by constraints implications (s13207).

number of implied signal values for the same benchmark circuit (s13207). Each constant PI produces on average 9.3 implied signal values. The dependence between the number of constant PIs and the number implied signal values appears to be linear. Similar behavior has been observed in all measured circuits.

Finally, we have compared the static and dynamic filters by the number of implied

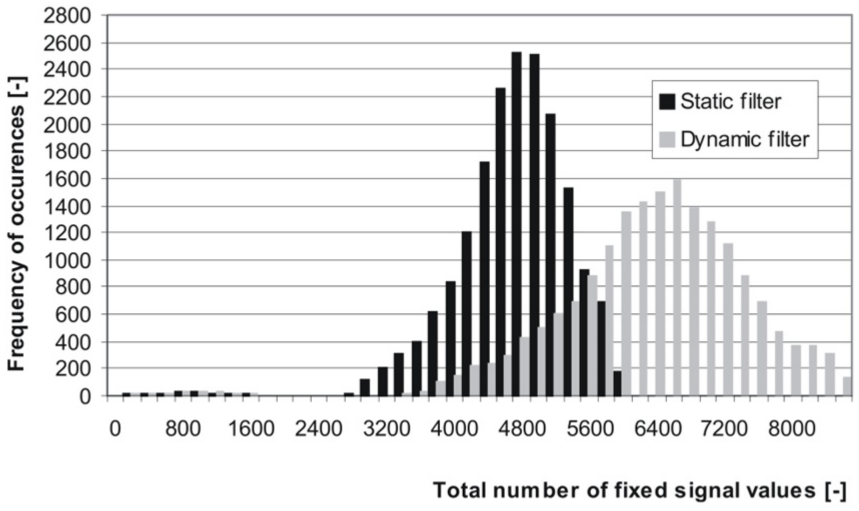

Figure 5.27: Frequency of values fixed by implications during CTPG (s13207).

signal values. The example for ISCAS'89 benchmark circuit s13027 is shown in Fig. 5.27. It seems that the distribution of the number of implied values is similar for both the static and dynamic filters. The only difference is their offset. The results from Tab. 5.11 show that the dynamic filter detects on average 10% more UNSAT CNF instances in comparison with the static filter. These observations confirm the assumption that more fixed signal values can detect more conflicts and increase the efficiency of the filter. Similar behavior has been observed in all measured circuits.

The implication filter seems to be a promising technique. The static filter can be used for any circuit and grants a significant speedup of the constrained test generation process by significantly decreasing the number of the unsatisfiable CNFs generated and solved. The dynamic filter is better for small circuits, because searching for dynamic implications is much more time-consuming.

Chapter 6

Dependability and Fault Classification

The principal way to improve the dependability of a circuit is to introduce redundancy. One possible strategy is to detect errors at output signals and take appropriate actions during circuit operation. This technique is called Concurrent Error Detection (CED, [163]).

Redundancy must be introduced with care, as redundant blocks are also prone to faults, and the point of diminishing return can be reached easily. Numerous schemes were devised to balance redundancy and dependability. To evaluate variant designs, we need to know how much dependability we get for a given investment into redundant circuits. Initially, dependability meant roughly what is today called robustness. In this thesis, we adhered to the original meaning from [164], where dependability parameters were introduced to quantify dependability.

The standard design flow is to design the circuit first, to construct its redundancy afterwards, and then to evaluate its dependability parameters. The underlying assumption is that the actual design, the technology used, and its resulting fault models influence the dependability parameters to a large degree. In many studies, the ubiquitous stuck-at models were used.

Recent Automatic Test Pattern Generation (ATPG) tools [165] and procedures based on solving the Satisfiability Problem (SAT) [22] permit analysis with a variety of fault models suitable for a particular circuit implementation technology. This in turn enables us to see the influence of technology on dependability. In other words, we ask whether there are circuits hard to make dependable or technologies hard to make dependable.

To study this question, we needed a simple framework. Currently, two approaches to robustness analysis and other tasks dealing with faults exist. The first one, represented by [166] and [31], transforms the task instance to an ATPG task instance. The ATPG tool then may or may not convert it internally to one or more SAT instances [1], [167]. The other approach, most notably represented by [22] and [24], converts an instance of the task to conceptual hardware (described in section 2.3) and then constructs SAT instances from that hardware.

Both approaches can be seen as special cases of a more general method, which can be summarized as follows. Firstly, transform the task instance into a piece of conceptual hardware together with assertions about the hardware. Secondly, use formal verification methods to prove or disprove the assertions (e.g. [165], [17]). If required, transform the assertions into conceptual hardware as well [2]. Thirdly, transform the answers back to the answers to the original task instance.

This is a powerful framework, which can even produce answers about sequential behavior in the presence of multiple faults [22], [23], [24]. For our study, combinational circuits (or full-scan circuits) were sufficient. Furthermore, only fault classification was required, without the need to analyze multiple fault impact.

Therefore, we present a simple framework for this limited situation, which constructs conceptual hardware representing the circuit and the assertions directly. We borrowed the term *miter* from ATPG [1], although the term *monitor* from [167] and other sources has a similar meaning.

We present a method to determine the value of an arbitrary Boolean formula over input vectors, error-free output vectors, and error-stricken output vectors of a combinational circuit. The formula can be quantified over all input vectors or their subset. We demonstrate our method to be compatible with methods used for multiple fault and sequential circuit modeling.

We present the method as an extension of SAT ATPG first in its general form. Then we review the class of dependable circuits studied and present their fault classification. We demonstrate application of the proposed method on this problem, and finally show and discuss classification results for two different implementation technologies.

6.1 Predicate Evaluation

This chapter deals with a predicate evaluation technique which is used in SAT-based ATPGs and suggests an extension, which allows evaluation of general predicates.

First, some basic terms about the SAT-Based ATPGs (see section 2.4, section 2.5) are repeated to remind some basic terms, which are important for further explanation.

6.1.1 Predicate Evaluation in The SAT-Based ATPGs

Let the circuit in question realize a Boolean function *F(x)* over input$x$. The circuit has n primary inputs and m primary outputs.

Denote $F_{flt}(x)$ the Boolean function characterizing the circuit with a given fault. The question whether the fault can be detected is answered by the predicate

$$\exists x, F(x) \neq F_{flt}(x) \tag{6.1}$$

In SAT-based ATPGs [1], [2], [167], this is understood as a circuit, see Fig. 6.1. The fault-free and faulty circuits provide *F(x)* and $F_{flt}(x)$, respectively. The predicate itself is also expressed as a conceptual hardware (circuit) called a *miter* [1]. The characteristic function of this conceptiual hardware (see section 2.4, section 2.5) is transformed to instance of the SAT problem (e.g. by Tseitin transformation, see section 2.4). If the SAT instance is unsatisfiable, the fault cannot be tested.

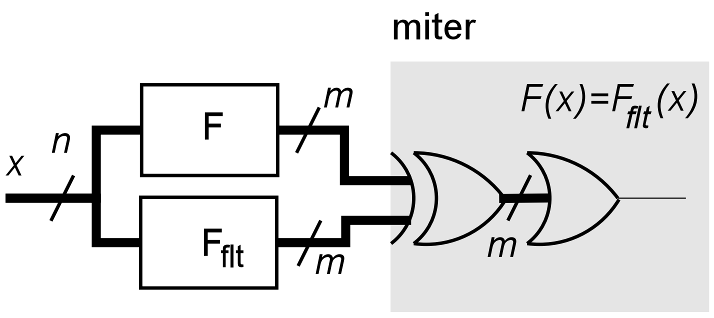

Figure 6.1: Circuit description of the ATPG SAT instance.

6.1.2 General Predicate

Let *x*, *F(x)* and $F_{flt}(x)$ have the same meaning as above. Let

$$\exists x, G(x, F(x), F_{flt}(x)) \tag{6.2}$$

be any Boolean predicate over *x*, *F(x)* and $F_{flt}(x)$. Then G can also be understood as a circuit, see Fig. 6.2. As it has the same role as in ATPG or model checking, we call it a *generalized miter*. Its characteristic function can be constructed as in the ATPG case, and the SAT instance is solved.

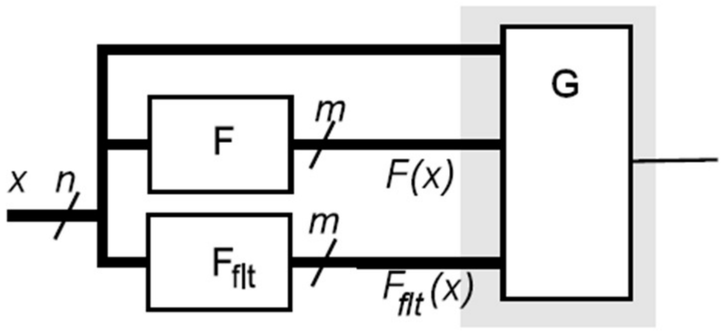

Figure 6.2: The generalized miter for predicate G.

A universally quantified predicate

$$\forall x, H(x, F(x), F_{flt}(x)) \tag{6.3}$$

can simply be converted to

$$\neg\exists x, \neg H(x, F(x), F_{flt}(x)) \tag{6.4}$$

The construction of $\neg H$ might seem dificult. When seen as a circuit, however, it suffices to add an inverter. This involves one more variable and two clauses in the SAT instance [45], which is tolerable. The predicate can be transformed to CNF by other methods as well; the above case illustrates the advantage of seeing it as a circuit. The dependency of the general predicate on x is useful in situations where not every input vector is admissible. Let A be the set of admissible input vectors and *a(x)* the predicate characterizing the set. Then

$$\exists x \in A, G(F(x), F_{flt}(x)) \tag{6.5}$$

becomes

$$\exists x, a(x) \wedge G(F(x), F_{flt}(x)) \tag{6.6}$$

This feature achieves the same effect as the input encoder in [166]. In the case solved there, code generator and detector for the codes in question are comparable in complexity. For other problems, however, to *produce* a vector may be more dificult than to *check* that vector.

The technique of generalized miters has a wider utilization than just single fault classification. For the sake of completeness, we demonstrate that it is compatible with procedures used for multiple fault reasoning and analysis of sequential circuits.

6.2 The Analyzed Architecture

6.2.1 The Structure of the Dependable Block

The CED strategy proposed in [163], [168] is used in this paper to illustrate principles of the proposed SAT-based predicate evaluation and for the experimental evaluation.

The digital circuit D to be secured by a CED code is supplemented with a predictor P and a checker E, see Fig. 6.3. The predictor can be understood as a copy of the functional circuit together with an encoder. The encoder transforms the vector at the primary outputs of the circuit into the redundancy bits of a selected error detection code. The primary outputs (POs) of the circuit to be secured and the predictor outputs form the codeword whose validity is verified by the checker. The original primary outputs *D(x)* together with the checker output form the global output *F(x)*, which is *m + 1* bits wide.

Any fault in the functional logic D either does not alter the output for a given input vector, or should be detected by the checker. Faults in the predictor and checker either do not affect the operation, or cause false alarms. This architecture can be apprehended as a kind of modification of the well-known duplex scheme (MDS architecture) [169], [4].

For the purpose of this study, single parity is used as the error detection code [4]. Thus, the predictor is constructed as a copy of the original circuit supplemented with a XOR tree at its outputs, $k = 1$ in Fig. 6.3.

The single parity code offers a low area overhead, however its error detection capabilities are limited. Therefore, the fault coverage can also be lower than in the case of the duplex system, and must be analyzed.

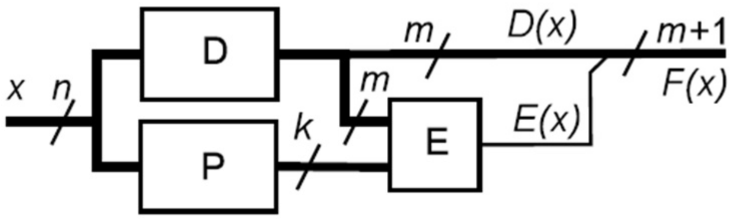

Figure 6.3: Basic concurrent error detection (CED) scheme.

6.2.2 Fault Classification and Dependability Parameters

There are three basic dependability parameters in the field of CED (Concurrent Error Detection) [163], [164]:

Fault security (FS) - probability that the erroneous outputs produced for a modeled fault do not belong to the output code-words.

Self-testing property (ST) - probability that an input vector occurring during normal operation produces an output vector which does not belong to the code when a modeled fault occurs.

Totally self-checking (TSC) - The FS and ST parameters of the circuit are equal to 100%. Totally Self-Checking property offers the highest level of protection.

The faults in the secured block cannot be classified only as detectable or undetectable, as for a common circuit. Their detectability by the checker must also be evaluated [22]. To compute these parameters, an approach based on a fault classification was presented in [22], [3], [4]. The faults are classified into four groups (A,B, C and D) based on their observability on primary outputs of the circuit and detectability by the checker.

Class A - these faults do not affect the circuit POs for any allowed input vector. This is the class of redundant (undetectable) faults. They have no impact to the FS property, but circuits with these faults cannot be ST.

Class B - these faults are detectable by at least one input vector and do not produce an incorrect code word (a valid code word, but incorrect) for other input vectors. They have no negative impact on the FS and ST properties, since if such a fault occurs, it is detected by the checker.

Class C - the faults that produce an incorrect codeword for at least one input vector and cannot be detected by any input vector. This is the class of faults, that can never be detected by the checker and that produce an erroneous output. The circuit with these faults is neither FS nor ST.

Class D - these faults cause at least one detectable and one undetectable error on the POs. They are detectable, but also may produce an incorrect output, which is not detected by the checker. They do not satisfy the FS property.

The FS property can be computed from the number of faults in these classes as:

$$FS = \frac{A+B}{A+B+C+D} \cdot 100 \quad [\%] \tag{6.7}$$

The ST property is computed in similar way as:

$$ST = \frac{B+D}{A+B+C+D} \cdot 100 \quad [\%] \tag{6.8}$$

where A, B, C, and D are the numbers of faults in the respective classes.

6.3 The SAT-Based Fault Classification Technique

To apply the SAT-based classification on the above outlined architecture, we must characterize the classes by binary predicates and apply the general scheme from Fig. 6.2.

6.3.1 Predicates

To compute the dependability parameters of the given architecture, each fault must be classified into one of the classes A, B, C, and D. Four classes need at least two binary predicates to distinguish. In this case, they are easy to derive from the specifications. In principle, the classes are defined by the ability of the fault to cause a detected or an undetected error, which can be formalized as follows:

- *J(x)* is true iff the input vector x gives an erroneous output *D(x)* of the faulty circuit and the error is detected (*E(x)* is *true*).
- *K(x)* is true iff the input vector x gives an erroneous output *D(x)* of the faulty circuit and the error is not detected (*E(x)* is *false*).

Then the given fault belongs to

- the class A, iff $\neg\exists x, J(x) \wedge \neg\exists x, K(x)$
- the class B, iff $\exists x, J(x) \wedge \neg\exists x, K(x)$
- the class C, iff $\neg\exists x, J(x) \wedge \exists x, K(x)$
- the class D, iff $\exists x, J(x) \wedge \exists x, K(x)$

Hence, two SAT instances must be solved to classify a fault.

6.3.2 Generalized Miters

To construct a miter for the J and K predicates, we have to apply the general process leading from the circuit in Fig. 6.1 to the circuit in Fig. 6.2 on the discussed architecture. The output *F(x)* is in our case decomposed into *D(x)* and *E(x)*, giving the circuit in Fig. 6.4.

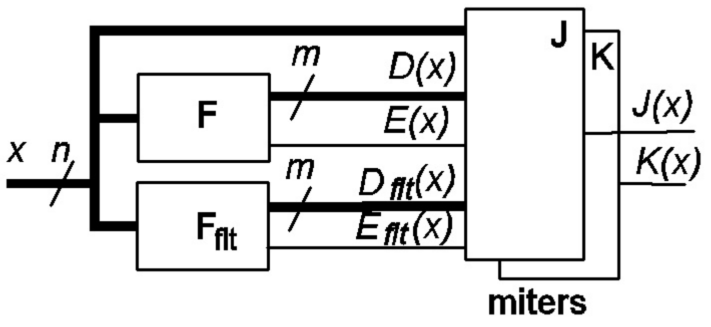

Figure 6.4: The general circuit J and K evaluation.

Bringing in the internal structure of F and F_{flt} from Fig. 6.3, we obtain the circuit in Fig. 6.5. The actual predicates apply to all input vectors x, therefore x does not enter into the miter circuits. Furthermore, we are interested in faults in the secured circuit D only, not in the predictor or checker. Therefore, we can omit $E_{flt}(x)$ from the miters and, therefore, D_{flt} and E_{flt} from the circuit. The final optimized circuit is in Fig. 6.6.

Using *D(x)*, $D_{flt}(x)$ and *E(x)*, we can implement the miters as

$$J(x) \equiv D(x) \oplus D_{flt}(x) \wedge E(x) \tag{6.9}$$

$$K(x) \equiv D(x) \oplus D_{flt}(x) \wedge \neg E(x) \tag{6.10}$$

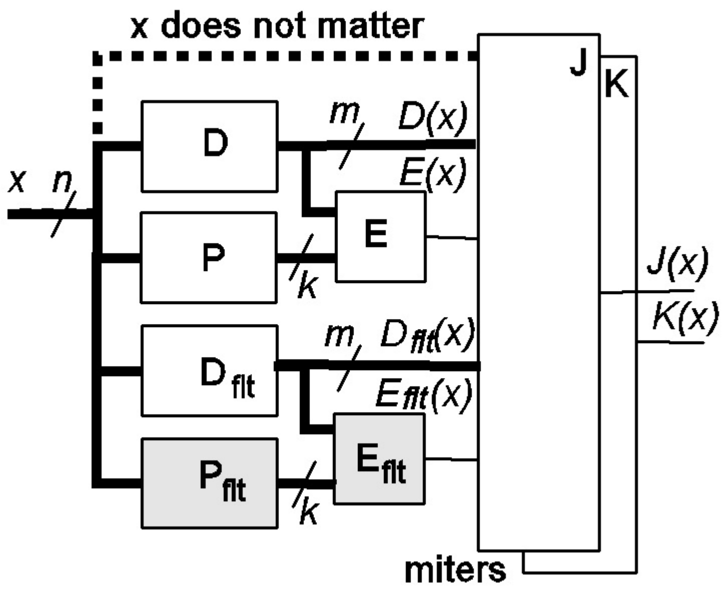

Figure 6.5: The unoptimized circuit for J and K.

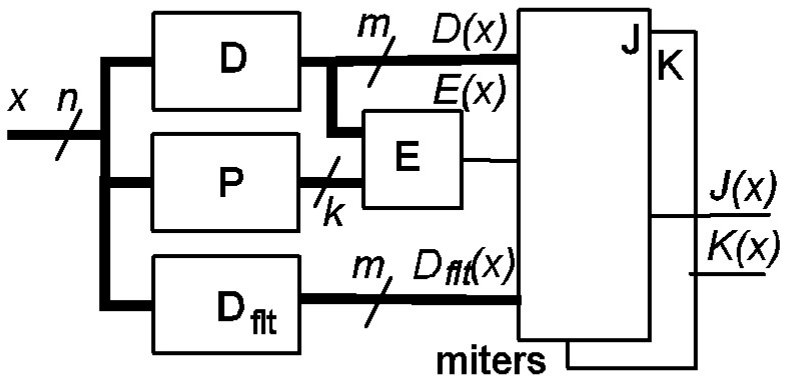

Figure 6.6: The optimized circuit for J and K.

6.4 Experimental Results

Experimental results can be divided into two parts.

The first part evaluates the efficiency and precision of our approach which is compared with exhaustive simulation of all patterns [170] on PIs of the evaluated circuit. This technique is practical for circuits having a small number of inputs only. Furthermore, Monte Carlo simulation is put in test as a representative of much faster dependability estimation. Here, the computed reliability parameters cannot provide reliable final evaluation.

The second part shows the robustnes of our framework for a set of benchmarks, implemented either as a network of gates with stuck-at faults, or implemented as a set of Look Up Tables (LUTs), considering Single Event Upset (SEU) in the LUT configuration memory as the primary fault mechanism.

All measurements were performed on a CPU Intel i5-2400 - 3,1GHz with 8GB RAM. The experiments have been performed on ISCAS'85 [87], ISCAS'89 [88], ITC'99 [89] and LGSynth [158] benchmark circuits.

6.4.1 Efficiency and Precision of Dependability Evaluation

The principal result presented in this section is the comparison of the proposed method with exhaustive simulation. Before that, however, we would like to present results that support our claim that Monte Carlo methods are insufficient for final evaluation.

6.4.1.1 Simulation Method Used for Comaprison

The simulator used in the experimental evaluation has been written specifically for this purpose. The signals in the circuit with an input vector applied were evaluated in topological order, in a serial fault simulation manner. The propagation of the examined fault was monitored only to the extent if odd or even number of outputs was affected. This method is limited to the case of the single-parity code, however, it is efficient and fast. On the other hand, the proposed method has not been restricted in a similar manner.

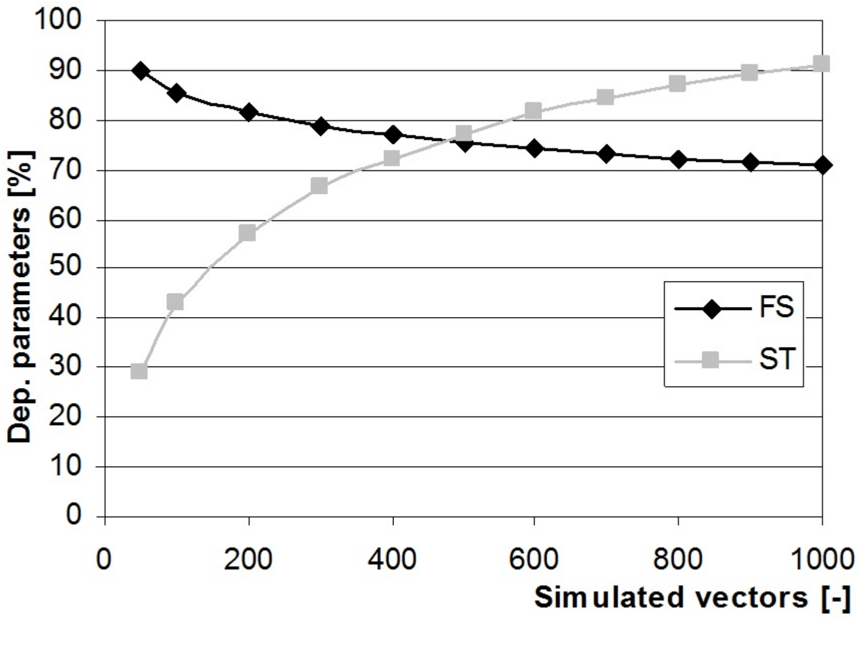

Figure 6.7: Dependance of *FS* and *ST* parameters on the number of simulated vectors (sao2).

6.4.1.2 Monte Carlo Simulation

Fig. 6.7 shows the dependence of the FS and ST parameters on the number of simulated vectors for *sao2* benchmark circuit [158] with ten PIs (2^{10} vectors for exhaustive simulation). The FS and ST parameters have been computed by Monte Carlo simulation for various numbers of random vectors. These measurements confirm the assumption that the number of simulated test patterns crucially affects the precision of dependability parameters computation mentioned in section 6.2.

6.4.1.3 Experimental Evaluation of the Proposed Method

A comparison of the SAT-based and exhaustive simulation for the fault classification is shown in Tab. 6.1. The first column of the table, *"Circ."*, represents the name of the benchmark circuit. The characteristics of the circuit are given by the following three columns. The number of primary inputs is shown in column *"PIs"*, the number of gates and faults are shown in columns *"Gat."* and *"Flts."*. The next two columns *"FS"* and *"ST"* present the computed fault security (FS) and self-testing (ST) property. The column *"Sim."* shows the time of the fault classification for simulation while the column *"SAT"* shows the fault classification time for the SAT-based approach. Finally, the column *"Acc."* shows the acceleration of the fault classification which can be achieved by the SAT-based approach.

The experimental results in Tab. 6.1. show that the SAT-based fault classification is much faster than the simulation based approach for all measured benchmarks, e.g., the fault classification time of the simulation-based algorithm for benchmark circuit *alu4* is 287210 seconds, while with a SAT-based algorithm it takes only 193 seconds.

The graph in Fig. 6.8 shows the acceleration of the fault classification gained by SAT algorithm as a function of the number of PIs. It seems that the acceleration grows exponentially with the number of PIs of the circuit (in the measured interval of PI counts). The feasibility of simulation time limited the experiments to circuits with up to 20 PIs.

The graph in Fig. 6.9 shows the time consumption of the SAT algorithm as a function of the circuit size expressed in equivalent gates [171]. It seems that the time consumption of the SAT algorithm grows almost linearly with the size of the circuit.

The number of simulated test vectors for the simulation based algorithm grows exponentially with the number of circuits PIs. The exponential gap between simulation and the SAT-based method suggests that the SAT solver runs in nearly polynomial time. This behavior has been expected, as the instances are almost 2-SAT ones [34], [168]. It is also confirmed for most of the circuits by experimental results in Fig. 6.9.

6.4.2 Experimental Technology Comparison

Following subsections discuss the influence of implementation technology on dependability parameters of combinational circuit. Moreover, it presents robustness of solution proposed in chapter 6 which is technology independent.

Table 6.1: Comparison of fault classification techniques

Circ.	PIs	Gat.	Flts.	FS	ST	Sim.	SAT	Acc.
	[-]	[-]	[-]	[%]	[%]	[s]	[s]	[-]
5xp1	7	161	422	84	98	6.8	0.5	12.5
dist	8	418	796	89	99	60.5	1.6	37.3
ex5p	8	1362	5506	51	73	2355.4	55.8	42.2
f51m	8	170	459	88	100	16.3	0.6	29.1
9sym	9	348	713	100	100	106.3	1.6	67.5
9symml	9	210	446	100	100	35.1	0.5	66.1
prom2	9	2322	5096	88	100	7010.2	56.2	124.8
sao2	10	669	549	71	92	94.94	0.9	104.9
x2	10	95	134	73	96	5.05	0.046	108
alu2	10	514	1132	44	89	438.8	3.9	113.0
newcond	11	119	149	93	100	17.05	0.109	156.14
cm85a	11	113	131	82	100	10.2	0.093	109
cm152a	11	69	56	100	100	1.88	0.032	60.5
alu1	12	82	109	83	100	15.1	0.1	121.2
br1	12	205	341	67	95	146	0.24	584.9
newpla	12	105	145	66	100	23.38	0.064	374.7
alu4	14	3540	7055	87	91	287210	193	1488.3
amd	14	435	842	75	97	4059.2	1.981	2048.8
cm162a	14	95	166	72	96	118.6	0.1	950.5
alcom	15	120	319	91	95	804.1	0.5	1718.1
b12	15	694	2109	93	47	51311.5	7.4	6924.6
gary	15	517	1059	80	98	13427.8	3.1	4369.3
in0	15	517	1059	80	98	13392.4	2.7	4905.6
intb	15	1261	1893	83	100	45669.5	24.4	1873.0
al2	16	141	400	85	98	2540.4	0.5	4789.5
ex7	16	206	297	77	100	1932.3	0.7	2880.6
prm1	16	160	176	73	94	626.2	0.124	5017.7
s298	17	200	287	66	98	3251	1.54	2105.4
s1196	32	863	1220	53	97	-	224.3	-
c1908	33	679	971	46	99	-	1261.8	-
jbp	36	771	1132	73	95	-	101.8	-
too_large	38	5477	12376	91	99	-	616.2	-
seq	41	2285	8914	71	94	-	117.2	-
c1355	41	412	882	80	100	-	2525.5	-
s953	45	690	1028	51	96	-	25.3	-
x1	51	974	2543	89	98	-	432.9	-

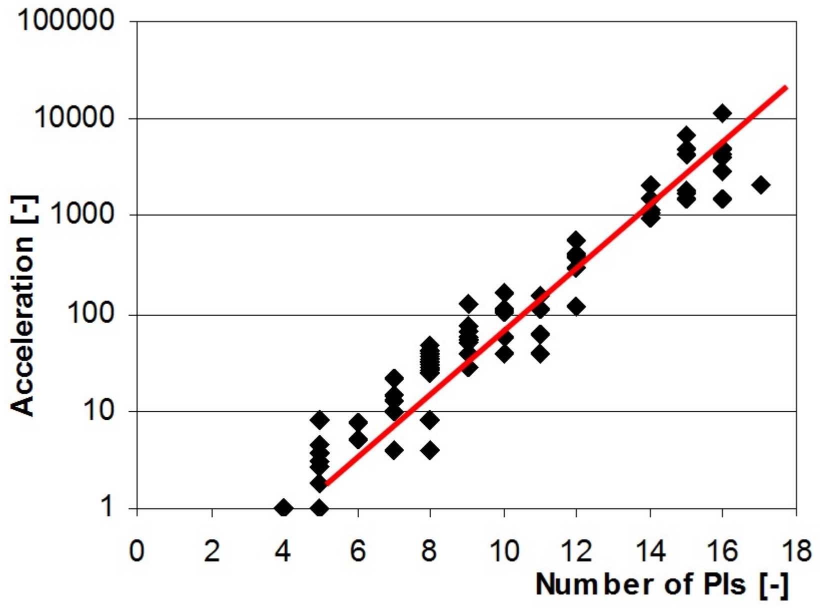

Figure 6.8: Dependance of the acceleration on the number of PIs.

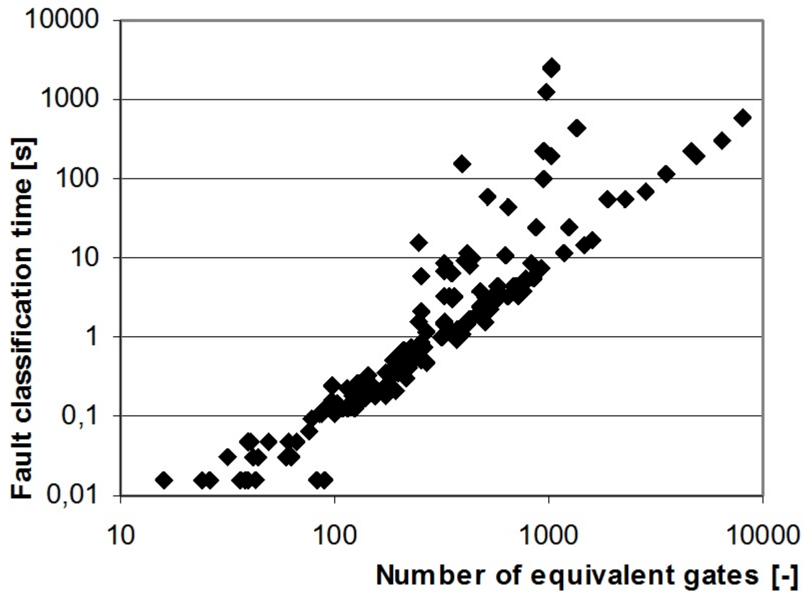

Figure 6.9: Dependance of the fault classification time on the size of the circuit.

6.4.2.1 Technology Comparison

Using the above described framework, we compared robustness of a set of benchmarks, implemented either as a network of gates with stuck-at faults, or implemented as a network of Look Up Tables (LUTs), considering Single Event Upset (SEU) in the LUT configuration memory as the primary fault mechanism.

The experiments have been performed on 67 ISCAS85 [87], ISCAS89 [88]], ITC99 [89] and LGSynth [158] benchmark circuits.

For the stuck-at faults, the original structural description was used. The fault lists were generated by Atalanta [147] and were free from dominated faults.

The LUT implementations were synthesized by ABC [172] using the command sequence *strash; dch; if; lutpack* as recommended by the ABC authors. Note that this procedure builds the implementation out of LUTs with 4 inputs or less.

The gate implementation was also produced by ABC with the command sequence *strash; dch; map.* The target gate library was the MCNC library [96].

Both implementations of a circuit have obviously different numbers of possible faults. To compare them in a practically relevant manner, we decided to count points of vulnerability, that is, the number of faults which can cause dysfunction of the circuit. The coeficients FS and ST, which indicate the distance to the Totally Self Checking goal, are of minor importance here. The metrics used were Not Fail Safe

$$NFS = C + D \tag{6.11}$$

and Not Self-Testing (NST)

$$NST = C \tag{6.12}$$

Tab. 6.2 shows the number of faults classified by the above described method.

The aggregate number of faults by category is given in Tab. 6.3, and the statistical properties are summarized in Tab. 6.4, using standard correlation and least square linear regression.

6.4.2.2 Total Number of Faults

The total numbers of faults in both technologies are tightly correlated. The circuit itself has the strongest inuence. The technology contributes only a constant coeficient, as the regression figure in Tab. 6.4 tells us that the LUT implementations has - almost uniformly - twice the number of total faults. The trend is most apparent in large circuits with more faults, as can be seen in Fig. 6.10.

This comparison, however, is influenced by the construction of the fault list for gates. A single stuck-at fault there can represent more than one dominated fault and hence more than one point of vulnerability, whereas dominance of SEU faults is unlikely and has not been considered.

Table 6.2: Detailed fault counts in two implementations

Circuit	**Gates, S@**						**LUTs, SEU**					
	Faults	**A**	**B**	**NST = C**	**D**	**NFS**	**Faults**	**A**	**B**	**NST = C**	**D**	**NFS**
5xp1	422	0	353	7	62	69	538	83	431	4	20	24
9symml	446	0	446	0	0	0	1036	275	761	0	0	0
9sym	713	0	713	0	0	0	1440	409	1031	0	0	0
al2	400	0	338	8	54	62	736	0	692	0	44	44
alcom	319	0	291	16	12	28	560	0	556	0	4	4
alu1	109	0	91	0	18	18	120	0	120	0	0	0
alu2	1132	117	383	9	623	632	1864	680	555	9	620	629
amd	842	0	632	23	187	210	2084	371	1546	16	151	167
b1	37	6	25	0	6	6	16	0	16	0	0	0
b9	366	2	205	45	114	159	560	6	412	64	78	142
br1	341	0	230	17	94	111	792	120	558	57	57	114
br2	296	0	185	20	91	111	564	58	406	18	82	100
c1355	882	0	704	0	178	178	1088	2	944	14	128	142
c17	22	0	12	0	10	10	32	0	32	0	0	0
c1908	971	5	437	0	529	529	1252	118	614	6	514	520
c432	553	8	82	0	463	463	1088	128	222	17	721	738
c499	882	0	704	0	178	178	1088	2	944	14	128	142
c8	636	56	489	0	91	91	454	21	401	0	32	32
cc	219	13	172	4	30	34	328	24	296	0	8	8
chkn	918	0	806	0	112	112	1924	269	1607	0	48	48
cht	669	39	604	0	26	26	588	0	584	0	4	4
clip	1108	27	964	3	114	117	1068	205	793	8	62	70
clpl	38	0	14	0	24	24	84	0	52	0	32	32
cm138a	74	0	68	6	0	6	96	0	96	0	0	0
cm150a	245	23	222	0	0	0	160	8	152	0	0	0
cm152a	56	0	56	0	0	0	72	4	68	0	0	0
cm162a	166	6	113	0	47	47	172	18	130	0	24	24
cm163a	159	4	115	0	40	40	160	4	156	0	0	0
cm42a	76	0	68	2	6	8	160	0	160	0	0	0
cm82a	60	0	17	0	43	43	32	0	24	0	8	8
cm85a	131	0	107	0	24	24	148	0	148	0	0	0
cmb	141	6	92	0	43	43	228	22	182	0	24	24
con1	51	0	43	0	8	8	68	2	66	0	0	0
count	379	0	265	4	110	114	520	24	432	0	64	64
cu	164	6	100	27	31	58	208	11	159	28	10	38
dc1	120	0	85	7	28	35	112	0	112	0	0	0
dc2	255	0	201	3	51	54	422	49	361	4	8	12
decod	130	0	122	8	0	8	192	0	192	0	0	0
dist	796	0	712	4	80	84	2240	552	1686	0	2	2
duke2	1303	1	609	13	2561	693	2476	482	914	264	816	1080
ex5	940	0	697	29	214	243	2768	917	1662	18	171	189
ex7	297	0	229	0	68	68	412	26	362	0	24	24
f51m	459	0	402	0	57	57	570	100	466	0	4	4
frg1	1049	0	1041	0	8	8	1248	118	1130	0	0	0
gary	1059	0	850	17	192	209	2508	445	1936	43	84	127
i1	129	1	89	2	37	39	162	0	138	0	24	24
ibm	492	0	354	0	138	138	1080	36	933	0	111	111
in0	1059	0	850	17	192	209	2416	415	1813	59	129	188
in2	1002	0	757	56	189	245	2060	289	1512	109	150	259
in4	1013	0	696	22	295	317	1928	230	1431	20	247	267
in5	802	0	549	16	237	253	1972	221	1539	17	195	212
in6	767	0	548	26	193	219	1384	159	1077	20	128	148
in7	311	0	167	12	132	144	668	75	441	57	95	152
jbp	1132	0	833	55	244	299	2222	168	1826	74	154	228
lal	414	0	295	11	108	119	374	15	279	0	80	80
ldd	278	9	164	45	60	105	404	71	238	42	53	95
luc	621	0	430	55	136	191	1192	182	819	108	83	191
m1	195	0	144	8	43	51	302	23	274	2	3	5
m2	543	0	442	7	94	101	956	170	729	16	41	57
m3	630	0	534	9	87	96	1546	340	1176	4	26	30
m4	973	0	844	8	121	129	2752	647	2044	14	47	61
majority	39	0	39	0	0	0	20	0	20	0	0	0
max46	380	0	380	0	0	0	744	110	634	0	0	0
max512	891	0	792	0	99	99	2788	716	2042	4	26	30
misex1	161	0	105	13	43	56	248	21	205	13	9	22
misex2	294	0	237	0	57	57	496	28	448	0	20	20
mlp4	694	0	590	12	92	104	1816	366	1437	0	13	13

Table 6.3: Total fault counts by category

	Faults	*A*	*B*	*NST = C*	*D*	*NFS*
gates	33251	329	24933	765	7224	7989
LUTs	61806	9835	45222	1143	5606	6749

Table 6.4: Statistical properties of fault numbers

Quantity	*Correlation*	*Lin. Regression*
Total faults	0.894	2.0
A	0.180	2.2
B	0.892	1.8
NST = C	0.934	1.77
D	0.947	1.15
NFS	0.949	0.73

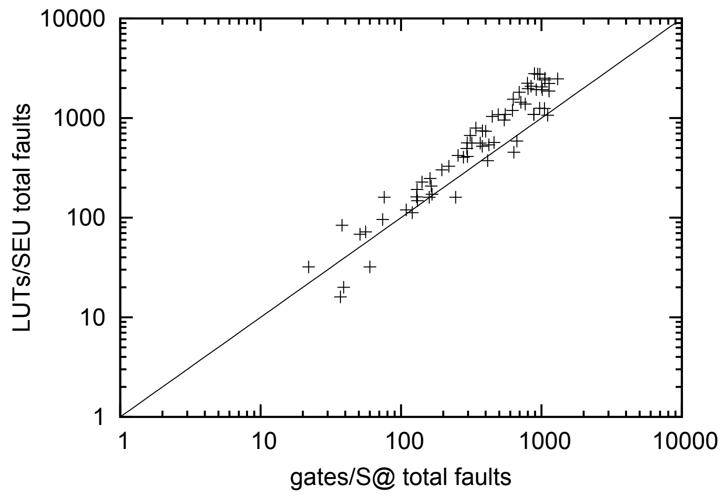

Figure 6.10: Total faults; $x = y$ provided for reference.

6.4.2.3 A-Class Faults

A-class faults are caused by redundancy introduced during synthesis. From the dependability point of view, this is a kind of noise in the implementation. From Fig. 6.11 it can be seen that the values are indeed uncorrelated. 4.5% of the circuits have A faults only in gate implementation, 57% only in LUT implementation, and 21% in both.

Note that A-class faults might also be introduced by not fully exploiting the 4-LUTs; there are many LUTs having less than 4 actual inputs in the synthesized designs. However, these faults are not considered in our computations, since they are not A-class faults,

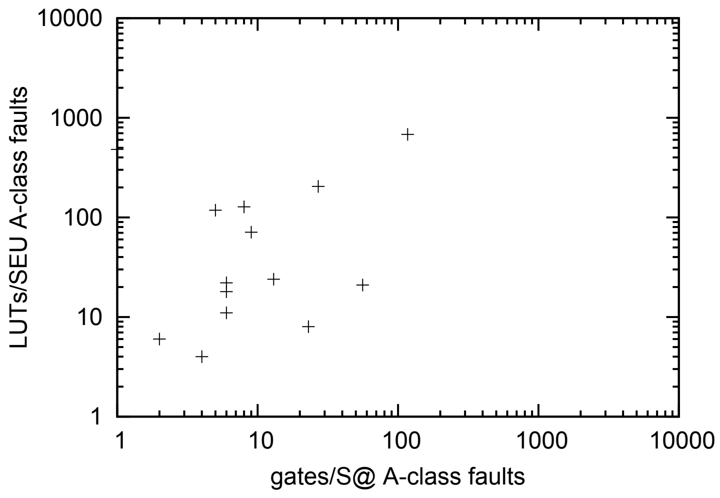

Figure 6.11: A faults.

indeed. If *any number* of SEUs in the unused parts of LUTs occurs, the dependability is not affected at all.

6.4.2.4 Points of Vulnerability

The values *NFS*, *NST*, which give the number of points of vulnerability, are correlated very tightly between the two implementations. Fig. 6.12 compares the numbers of Not Self-Testing faults, and Fig. 6.13 compares the final vulnerability indicator, the Not Fault Secure faults. Tab. 6.3 and Tab. 6.4 together with Fig. 6.12, Fig. 6.13, Fig. 6.14 and Fig. 6.15 indicate that while the LUT technology has bigger numbers of faults, most of them are in the B category, and that the number of faults in the C and D categories move in opposite directions, so that the number of vulnerable points - the NFS faults - remains almost constant.

It follows that the dependability, or, more precisely, the ability to become dependable using the MDS architecture, does not depend on architecture and fault model. Rather, it is a property of the circuit itself.

6.4.3 Extension to Multiple Faults

In [42] and [22], a technique to model multiple faults in a circuit is presented. The presence or absence of each fault is modeled by an associated fault predicate. The actual value of signals in question is modeled by additional primary inputs, which in turn are expressed by free variables in the SAT instance.

The structure of the conceptual hardware including the generalized miter remains the same as in Fig. 6.2, except the faulted copy of F is controlled by the fault predicates and additional inputs.

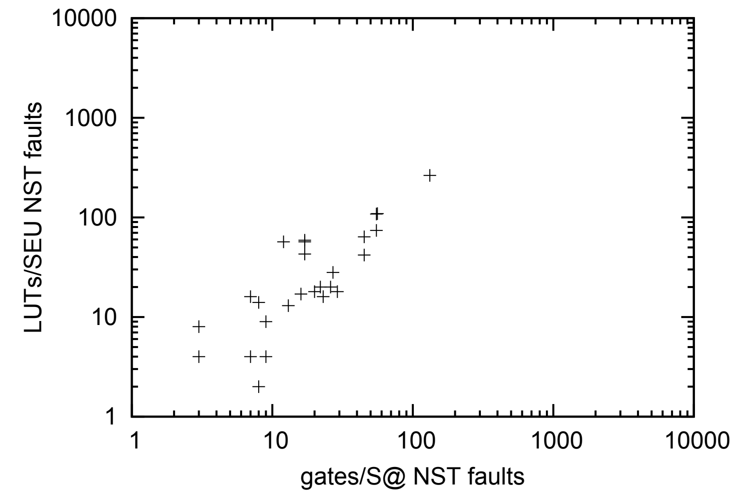

Figure 6.12: NST = C faults.

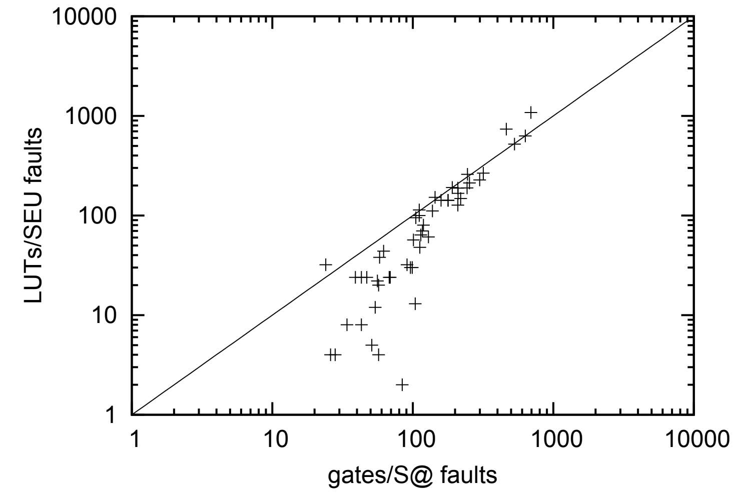

Figure 6.13: NFS faults; $x = y$ provided for reference.

6.4.4 Extension to Sequential Circuits

[22] uses time-frame unrolling to reason about sequential circuits. The circuit is divided into a state register and combinational logic. For each time step, a stage is created comprising a copy of the fault-free and faulty logic. The outputs of both copies in each stage are compared by a miter. The value of state variables passes from stage to stage.

The miter can be replaced by an generalized miter, as in Fig. 6.16. The construction of the generalized miter depends on the way it is specified. If the miter is combinational, then each stage contains one copy of the miter, and all outputs of those stage miters are

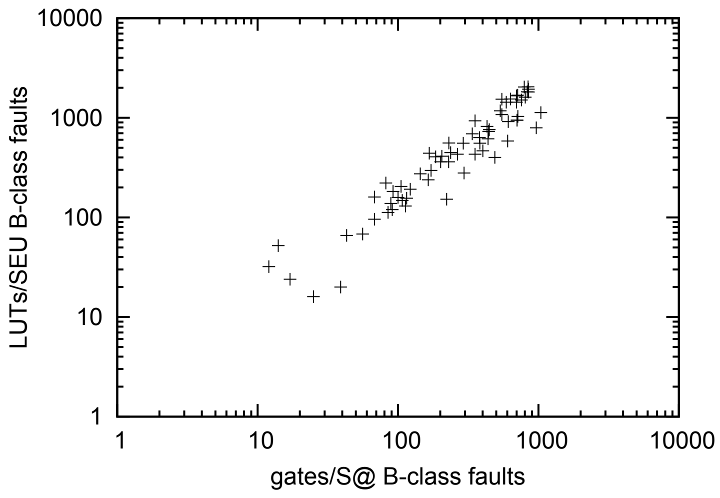

Figure 6.14: B-class.

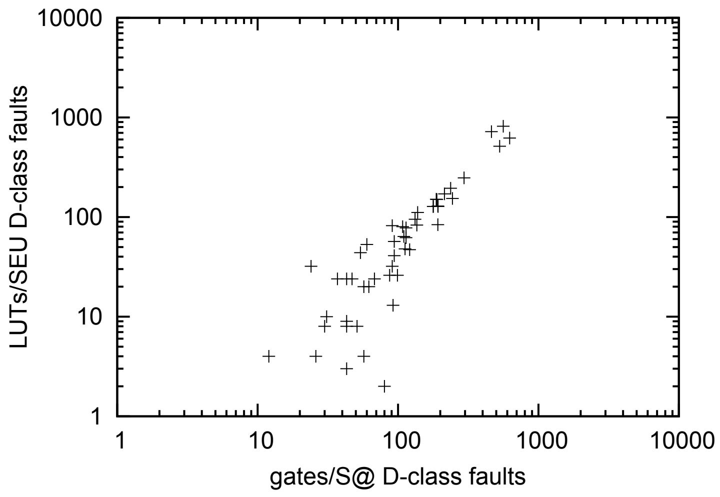

Figure 6.15: D-class.

combined by e.g. an AND gate. If the miter is described as a state machine, then it is unrolled similarly to F and the state of the miter is passed from state to state.

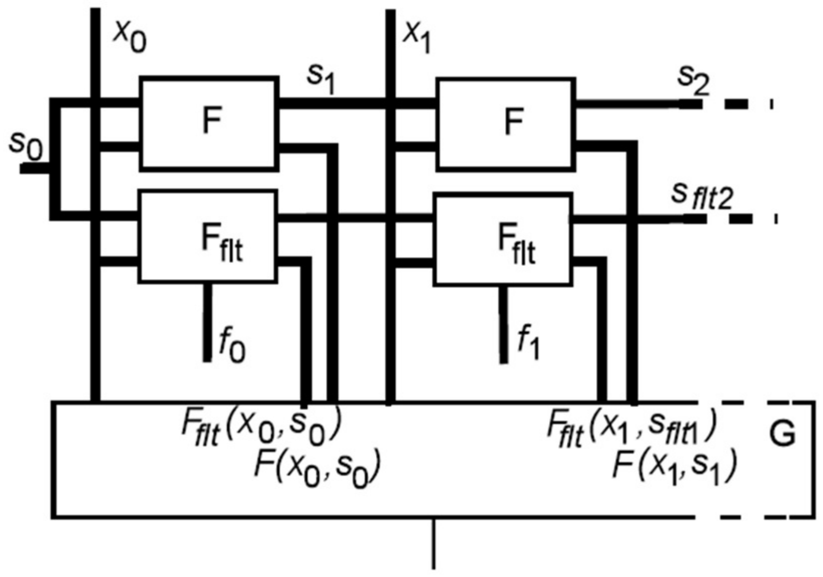

Figure 6.16: The generalized miter for predicate G in unrolled sequential circuit; x_i, s_i and f_i are inputs, states, and fault predicates, respectively, in time t_i.

CHAPTER 7

Discussion

This thesis deals with application of implicit representations of vector sets in the field of digital circuits testing and dependability evaluation. Implicit representations of a vector set can represent a large amount of data with significantly lower memory requirements than explicit representations. Moreover, chapter 3 shows that the application of implicit representations of a vector set caused a significant breakthrough for many problems. However, in some cases, their application can cause more harm then good.

Pilot examples introduced in previous chapters show examples of two problems which seem to be very appropriate to application of implicit representations of the vector set. Their application should cause significant improvement in both the processing time and efficiency, but our experimental results show, that only the second pilot example (fault classification) reaches significantly better results (precision and time-consumption). The first pilot example (serial compression) reaches results comparable with state-of-the-art tools, but its scalability is limited.

Generally, it can be concluded that algorithms, which accomodate the implicit representation of vector sets must deal with generation of the implicit representation and the way of its processing. Summarization in chapter 3 suggests, that for implicit representations of a vector set, the generation and processing are mutually orthogonal. It means that the implicit representations, which are easy to generate, are much harder to process and vice versa.

Our research was focused primarily on the implicit representation of the vector set in CNF, which is widelly used in state-of-the-art EDA tools. It is highly scalable, because both its size and generation time grows linearly with the circuit size. Moreover, its processing (simplification, constraining, SAT solving) is very fast in most cases. Despite of that, there are problems which cause troubles.

Now, let us discuss the two case studies:

1. The first Case study deals with a serial test compression performed by SAT-Compress, which is a very simple greedy algorithm. This algorithm should quickly preform the test patterns compression for any circuit, because both CNF generation and the SAT

solving are performed in nearly polynomial time in most cases. The compression ratio is high as expected, but our experimental results show that its scalabily is limited and its application on large industrial circuits is impossible.

The performance overhead is caused by the huge number of unsatisfiable instances (80% - 99% of solved CNFs in most cases) which are implicitly produced by constraints application. Furthermore, our measurement confirmed previous observations [2] that CNF generation is much more time-consuming than its solving (80% of time for the SAT-Compress). Thus, we tried to eliminate this weeknes by application of UNSAT filters (see subsection 5.4.2) or another kind of CNF processing like CNF storing in memory (see subsection 5.4.1). Our experimental results proved, that we can easily eliminate about 50% of UNSAT instances and beside that SAT-Compress with CNF storing in memory can perform the compression 1.34-times faster than SAT-Compress with CNF generation on-the-fly. Unfortunately, scalability of CNF storing is strictly limited by memory requirements. Moreover, the acceleration of the algorithm is still insufficient for industrial circuits.

Another interesting part of our research explores the influence of DCs in test patterns on compression ratio. Algorithms' processing pregenerated test pattern set prefers test patterns with as many DCs as possible [73], [149]. These patterns are easier to overlap, which consequently produces shorter bitstream. However, our experimental results show, that injection of DC bits in test patterns can cause degradation of the compression ratio for an algorithm like SAT-Compress which utilizes implicit representation of the vector set and does not depend on pregenerated test (see section 5.2). This behavior can be expected, because SAT-Compress can choose any test pattern from the test set of each fault, while algorithms processing pregenerated test set have at most one test pattern per fault. Our experimental results show that the improvement of compression ratio implicitly caused reduction of the number of CNF instances to be processed, which caused acceleration of the algorithm proportional to the improvement of the compression ratio.

Our further research on "influences" in serial compression (see chapter 5) suggests that the main problem is the generation of sequence of vectors which are closely tied to each other. Here, compression results are highly influenced by many factors like initial conditions (initial test pattern, order of PIs,) or processing options (DCs injection), which can significantly influence both the compression ratio and time. Unfortunatelly, it is nearly impossible to find optimal combination of these initial and processing conditions to get optimal results in reasonable time.

Explicit techniques like CNF filtering, CNF storing, etc., can reduce *"consequences"* and significantly accelerate the compression algorithm, but they cannot overcome troubles caused by the *"problem itself"* (sequence generation).

Implicit techniques like DC processing (injection) seems to be much more beneficial for this kind of problems (sequence generation), but they represent just one small piece of puzzle which cannot save the day.

Finally, we can conclude that our first expectations about *"suitability"* of application of implicit representation was not fulfilled. The best compression ratio cannot be achieved by a *"simple"* selection of the best pattern in the best order, but it significantly relies on combination of many other factors and selection of suitable pattern has minor influence on the compression ratio and generation time. Such class of problems, where a sequence of vectors is generated, is not very suitable for application of implicit representation.

2. The second case study deals with the fault classification which is crucial for precise dependability evaluation. Here, the main goal is to decide if there exists any pattern which fulfills some prediction. Similar decision problems should be very suitable for application of implicit representation of a vector set. Current Boolean proof engines are highly efficient and any decision problem can be easily described by conceptual hardware.

   This assumption was confirmed by our experimental results (see section 6.4). The efficiency of the proposed algorithm is very high, while the precision of results is preserved. Furthermore, these techniques for application of an implicit representation can be easily extended to optimization problems and simply solved by any PBO solver or applied to similar problems like those described in subsection 6.4.3 and subsection 6.4.4.

Chapter 8

Conclusions

8.1 Summary

The main aim of this thesis is to discuss the application of implicit representations of vector sets in serial compression and dependability evaluation.

First, the application of implicit representations of the vectors set in different fields of digital system design, testing and verification has been briefly summarized (see. chapter 3). Here, the problems where application of implicit representation caused a significant breakthrough where highlighted.

The implicit representation of the vector set as a SAT instance in CNF has been closely studied (see chapter 4). It is easy to handle and its application can yield in a great robustness and performance improvement.

An extensive research on properties of SAT instances produced by Tseitin transformation from a digital circuit confirms that SAT instances produced this way are *"easy to be solved"*. Moreover, we have found that these SAT instances significantly differ from randomly generated ones of equal parameters, particularly in terms of their satisfiability. ATPG SAT instances are mostly satisfiable, even though random instances of the same parameters should not be. Further research shows, that SAT instances produced by the Tseitin transformation involve a great number of redundancies. On average 60% of variables and 65% of clauses can be removed by application of solution preserving reductions.

The SAT-Compress algorithm for serial test patterns compression has been introduced (see chapter 5). This simple greedy algorithm can reach high compression ratio (86% - 90% on average) comparable with similar state-of-the-art compression algorithms.

Extending techniques of speedup for SAT-Compress have been discussed and evaluated by experimental results. The differences between the CNFs processing on-the-fly, processing of stored CNFs or reduced CNFs have been discussed and shown on a set of ISCAS'85 and '89 benchmark circuits. It can be concluded that even if a generation of the CNFs on-the-fly can be time-consuming, it is still the best technique of CNF processing in a general case.

Techniques of filtering of unsatisfiable CNFs based on static and dynamic implications have been presented and their properties have been shown on the SAT-Compress algorithm. Our experimental evaluation proved that it can be a powerfull technique for speedup of SAT-based constrained test patterns generation.

Furthermore, the role of don't cares in the compressed test generation was studied. Several techniques for obtaining don't cares in the test are proposed, both uninformed and informed ones. We have shown that don't cares obtained in an uninformed way cannot be efficiently exploited in test compression and sometimes they even have disturbing effects.

The observations resulted in an efficient enhancement of the SAT Compress ATPG algorithm, the Coverage Preserving Don't Care Injection technique (CPDCI). Basically, the SAT Compress algorithm gradually constructs compressed test patterns by repetitively solving the SAT problem for instances constrained by patterns generated in previous steps. The CPDCI technique significantly alleviates these constraints by substituting defined values by don't cares, without any loss of the fault coverage in each step. This is accomplished by a procedure based on a symbolic fault simulation. Less constrained SAT instances allow reaching better results, both in test bitstream size (by 46% on average) and test generation time (by 35% on average). We see that even though the fault simulation imposes some computational overhead, the resulting run time is significantly reduced, because of shorter bitstreams generated.

A method for proving arbitrary predicates quantified over input vector of a combinational circuit has been presented (see chapter 6). The method combines elements from SAT ATPG and SAT-based property checking. The Modified Duplex System architecture, which requires classification into four classes, has been selected for demonstration of the method. Experimental evaluation on a number of benchmarks proved that the method runs in a time close to polynomial and is practical even for circuits with a large number of inputs, where exhaustive simulation is bound to fail.

Furthermore, a set of benchmark circuits was constructed using the MDS redundancy architecture. The circuits were implemented both in gates and LUTs. Their self-checking characteristics were evaluated by the described method under the stuck-at and single event upset fault models, respectively. The characteristics were found to be correlated, which suggests that the ability to become dependable under the MDS scheme is an intrinsic property of the circuit itself.

Finally, the application of implicit representations in the field of serial compression and dependability evaluation has been discussed (see chapter 7).

8.2 Contributions

The contributions of this thesis are summarized in the following list:

- Application of implicit representations of the vectors set in different fields of digital system design, testing, and verification has been summarized (see. chapter 3).
- An extensive research on properties of SAT instances produced by Tseitin transformation from combinational circuits has been performed (see chapter 4).

- The SAT-Compress [A.10], [A.9], [A.17], [A.5] algorithm has been introduced as a pilot example for serial compression (see chapter 5). A simple and easy to implement SAT-based compression algorithm, which is able to reach the compression ratio comparable with state-of-the-art algorithms (86-90% on average) has been proposed.
- Techniques to increase the efficiency of SAT-based serial compression have been proposed and discussed [A.5], [A.1], [A.2], [A.7] (see section 5.3 and section 5.4).
- A novel and original method to convert a conceptual hardware (miter) to a Pseudo Boolean Optimization (PBO) instance was proposed [A.1], [A.2] (see section 5.3 and subsection 5.3.2).
- The fault classification [A.6], [A.8], [A.12] tool was introduced as a pilot example for a fast and precize fault classification (see chapter 6). This tool utilizes an implicit representation of vectors for proving arbitrary predicates quantified over an input vector. This method combines features from SAT ATPG and SAT based property checking.

8.3 Future Work

The author of doctoral thesis suggests exploring the following:

- An advanced application of implicit representations of vector sets in fields like multiple fault testing and diagnosis for both combinational and sequential circuits (see subsection 6.4.3 and subsection 6.4.4). Such problems seem to be suitable for application of implicit representations.
- Further research on PBO encoding and transformation of decision problems to optimization problems. Here, for example, application of implicit representation in power-aware testing to generate test set with custom properties would be interesting.
- Explore other promising implicit representations of the vector set like Structurally Synthetized Binnary Decision Diagrams (SSBDD) [173], [174] and their modifications like Structurally Synthetized Multiple Input BDDs (SSMIBDD) [175], [176] etc.

Bibliography

[1] Larrabee, T. Test pattern generation using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design*, 1992: pp. 4–15.

[2] Drechsler, R.; Eggersglüß, S.; Fey, G.; et al. *Test Pattern Generation Using Boolean Proof Engines*. Springer Publishing Company, Incorporated, first edition, 2009, ISBN 9048123593, 9789048123599.

[3] Dobias, R.; Kubalik, P.; Kubatova, H. Dependability computations for fault-tolerant system based on FPGA. In *Proceedings of the 12th IEEE International Conference on Electronics, Circuits and Systems*, Dec 2005, pp. 1–4, doi:10.1109/ICECS.2005.4633533.

[4] Kubalík, P.; Kubátová, H. Dependable Design Technique for System-on-chip. *J. Syst. Archit.*, volume 54, no. 3-4, Mar. 2008: pp. 452–464, ISSN 1383-7621, doi:10.1016/j.sysarc.2007.09.003. Available from: `http://dx.doi.org/10.1016/j.sysarc.2007.09.003`

[5] Dorsch, R.; Wunderlich, H. J. Tailoring ATPG for embedded testing. In *Proceedings of the International Test Conference*, 2001, ISSN 1089-3539, pp. 530–537, doi:10.1109/TEST.2001.966671.

[6] Novák, O.; Zahrádka, J.; Plíva, Z. COMPAS - Compressed Test Pattern Sequencer for Scan Based Circuits. In *Proceedings of the 5th European Dependable Computing Conference*, 2005, pp. 403–414, doi:10.1007/11408901_30. Available from: `http://dx.doi.org/10.1007/11408901_30`

[7] Su, C.; Hwang, K. A serial scan test vector compression methodology. In *Proceedings of the International Test Conference*, Oct 1993, pp. 981–988, doi:10.1109/TEST.1993.470601.

[8] Brayton, R. K.; Sangiovanni-Vincentelli, A. L.; McMullen, C. T.; et al. *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984, ISBN 0898381649.

[9] McGeer, P.; Sanghavi, J.; Brayton, R.; et al. ESPRESSO-SIGNATURE: A New Exact Minimizer for Logic Functions. In *Proceedings of the 30th international Design Automation Conference*, June 1993, ISSN 0738-100X, pp. 618–624, doi: 10.1109/DAC.1993.204022.

[10] Burch, J. R.; Clarke, E. M.; McMillan, K. L.; et al. Symbolic model checking: $10\hat{2}0$ states and beyond. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, Jun 1990, pp. 428–439, doi:10.1109/LICS.1990.113767.

[11] Schulz, M. H.; Fuchs, K.; Fink, F. Advanced automatic test pattern generation techniques for path delay faults. In *Proceedings of the 9th International Symposium on Fault-Tolerant Computing*, June 1989, pp. 44–51, doi:10.1109/FTCS.1989.105541.

[12] Roth, J. P. Diagnosis of Automata Failures: A Calculus and a Method. *IBM Journal of Research and Development*, volume 10, no. 4, July 1966: pp. 278–291, ISSN 0018-8646, doi:10.1147/rd.104.0278.

[13] Goel, P. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, volume C-30, no. 3, March 1981: pp. 215–222, ISSN 0018-9340, doi:10.1109/TC.1981.1675757.

[14] Armstrong, D. B. On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Nets. *IEEE Transactions on Electronic Computers*, volume EC-15, no. 1, Feb: pp. 66–73, ISSN 0367-7508, doi:10.1109/PGEC.1966.264376.

[15] Fujiwara, H.; Shimono, T. On the Acceleration of Test Generation Algorithms. *IEEE Transactions on Computers*, volume C-32, no. 12, Dec 1983: pp. 1137–1144, ISSN 0018-9340, doi:10.1109/TC.1983.1676174.

[16] Schulz, M. H.; Trischler, E.; Sarfert, T. M. SOCRATES: a highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 7, no. 1, Jan 1988: pp. 126–137, ISSN 0278-0070, doi:10.1109/43.3140.

[17] Biere, A.; Kunz, W. SAT and ATPG: Boolean engines for formal hardware verification. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov 2002, ISSN 1092-3152, pp. 782–785, doi:10.1109/ICCAD.2002.1167620.

[18] Burch, J. R.; Singhal, V. Tight integration of combinational verification methods. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov 1998, pp. 570–576, doi:10.1109/ICCAD.1998.144325.

[19] Eggersglüß, S.; Wille, R.; Drechsler, R. Improved SAT-based ATPG: More constraints, better compaction. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2013, ISSN 1092-3152, pp. 85–90, doi:10.1109/ICCAD.2013.6691102.

[20] Drechsler, R. BiTeS: A BDD Based Test Pattern Generator for Strong Robust Path Delay Faults. In *Proceedings of the Conference on European Design Automation*, EURO-DAC '94, Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, ISBN 0-89791-685-9, pp. 322–327. Available from: `http://dl.acm.org/citation.cfm?id=198174.198276`

[21] Eggersglüß, S.; Fey, G.; Glowatz, A.; et al. MONSOON: SAT-Based ATPG for Path Delay Faults Using Multiple-Valued Logics. *J. Electron. Test.*, volume 26, no. 3, June 2010: pp. 307–322, ISSN 0923-8174, doi:10.1007/s10836-010-5146-y. Available from: `http://dx.doi.org/10.1007/s10836-010-5146-y`

[22] Fey, G.; Drechsler, R. A Basis for Formal Robustness Checking. In *Proceedings of the 9th International Symposium on Quality Electronic Design*, March 2008, ISSN 1948-3287, pp. 784–789, doi:10.1109/ISQED.2008.4479838.

[23] Frehse, S.; Fey, G.; Suflow, A.; et al. Robustness Check for Multiple Faults Using Formal Techniques. In *Proceedings of the 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools*, Aug 2009, pp. 85–90, doi: 10.1109/DSD.2009.218.

[24] Fey, G.; Sulflow, A.; Frehse, S.; et al. Effective Robustness Analysis Using Bounded Model Checking Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 30, no. 8, Aug 2011: pp. 1239–1252, ISSN 0278-0070, doi:10.1109/TCAD.2011.2120950.

[25] Akers, S. B. Binary Decision Diagrams. *IEEE Trans. Comput.*, volume 27, no. 6, June 1978: pp. 509–516, ISSN 0018-9340, doi:10.1109/TC.1978.1675141. Available from: `http://dx.doi.org/10.1109/TC.1978.1675141`

[26] Bryant, R. E. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, volume C-35, no. 8, Aug 1986: pp. 677–691, ISSN 0018-9340, doi:10.1109/TC.1986.1676819.

[27] Drechsler, R.; Sieling, D. Binary decision diagrams in theory and practice. *STTT*, volume 3, no. 2, 2001: pp. 112–136, doi:10.1007/s100090100056. Available from: `http://dx.doi.org/10.1007/s100090100056`

[28] Fujita, M.; McGeer, P. C.; Yang, J. C.-Y. Multi-Terminal Binary Decision Diagrams: An Efficient DataStructure for Matrix Representation. *Form. Methods Syst. Des.*, volume 10, no. 2-3, Apr. 1997: pp. 149–169, ISSN 0925-9856, doi:10.1023/A:1008647823331. Available from: `http://dx.doi.org/10.1023/A:1008647823331`

[29] Kebschull, U.; Schubert, E.; Rosenstiel, W. Multilevel logic synthesis based on functional decision diagrams. In *Proceedings of the 3rd European Conference on Design Automation*, Mar 1992, pp. 43–47, doi:10.1109/EDAC.1992.205890.

[30] Schafer, L.; Dorsch, R.; Wunderlich, H. J. RESPIN++ - deterministic embedded test. In *Proceedings of the 7th IEEE EuropeanTest Workshop*, 2002, ISSN 1530-1877, pp. 37–44, doi:10.1109/ETW.2002.1029637.

[31] Hunger, M.; Hellebrand, S.; Czutro, A.; et al. ATPG-based grading of strong fault-secureness. In *Proceedings of the 15th IEEE International On-Line Testing Symposium*, June 2009, ISSN 1942-9398, pp. 269–274, doi:10.1109/IOLTS.2009.5196027.

[32] Girard, P.; Nicolici, N.; Wen, X. Power-Aware Testing and Test Strategies for Low Power Devices. Springer Publishing Company, Incorporated, 2009, ISBN 1441909273, 9781441909275.

[33] Stanion, T.; Bhattacharya, D. TSUNAMI: a path oriented scheme for algebraic test generation. In *Proceedings of the 21st International Symposium on Fault-Tolerant Computing*, June 1991, pp. 36–43, doi:10.1109/FTCS.1991.146630.

[34] Prasad, M. K.; Chong, P.; Keutzer, K. Why is ATPG easy? In *Proceedings of the 36th Design Automation Conference*, 1999, pp. 22–28, doi:10.1109/DAC.1999.781224.

[35] Jenicek, J.; Novak, O. Test Pattern Compression Based on Pattern Overlapping. In *Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems*, April 2007, pp. 1–6, doi:10.1109/DDECS.2007.4295250.

[36] Chloupek, M.; Novák, O. Test pattern compression based on pattern overlapping and broadcasting. In *Proceedings of the 10th International Workshop on Electronics, Control, Measurement and Signals*, June 2011, pp. 1–5, doi:10.1109/IWECMS.2011.5952372.

[37] Cook, S. A. The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, New York, NY, USA: ACM, 1971, pp. 151–158, doi:10.1145/800157.805047. Available from: `http://doi.acm.org/10.1145/800157.805047`

[38] Aspvall, B.; Plass, M. F.; Tarjan, R. E. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Inf. Process. Lett.*, volume 8, no. 3, 1979: pp. 121–123, erratum: Information Processing Letters 14(4): 195 (1982).

[39] Monasson, R.; Zecchina, R.; Kirkpatrick, S.; et al. 2+p-SAT: Relation of typical-case complexity to the nature of the phase transition. *Random Struct. Algorithms*, volume 15, no. 3-4, 1999: pp. 414–435. Available from: `http://onlinelibrary.wiley.com/doi/10.1002/(SICI)1098-2418(199910/12)15:3/4&lt;414::AID-RSA10&gt;3.0.CO;2-G/abstract`

[40] Kirsh, D. *Implicit and Explicit Representation.* John Wiley & Sons, Ltd, 2006, ISBN 9780470018866, doi:10.1002/0470018860.s00166. Available from: `http://dx.doi.org/10.1002/0470018860.s00166`

[41] Mishchenko, A.; Chatterjee, S.; Brayton, R. DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In *Proceedings of the 43rd ACM/IEEE Design Automation Conference*, 2006, ISSN 0738-100X, pp. 532–535, doi:10.1109/DAC.2006.229287.

[42] Smith, A.; Veneris, A.; Ali, M. F.; et al. Fault diagnosis and logic debugging using Boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 24, no. 10, Oct 2005: pp. 1606–1621, ISSN 0278-0070, doi:10.1109/TCAD.2005.852031.

[43] Garey, M. R.; Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* New York, NY, USA: W. H. Freeman & Co., 1990, ISBN 0716710455.

[44] Jackson, P.; Sheridan, D. Clause Form Conversions for Boolean Circuits. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, SAT'04, Berlin, Heidelberg: Springer-Verlag, 2005, ISBN 3-540-27829-X, 978-3-540-27829-0, pp. 183–198, doi:10.1007/11527695_15. Available from: `http://dx.doi.org/10.1007/11527695_15`

[45] Tseitin, G. S. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic, Part2*, 1962, pp. 115–125.

[46] Manolios, P.; Vroon, D. Efficient Circuit to CNF Conversion. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing*, SAT'07, Berlin, Heidelberg: Springer-Verlag, 2007, ISBN 978-3-540-72787-3, pp. 4–9. Available from: `http://dl.acm.org/citation.cfm?id=1768142.1768145`

[47] Stephan, P.; Brayton, R. K.; Sangiovanni-Vincentelli, A. L. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 15, no. 9, Sep 1996: pp. 1167–1176, ISSN 0278-0070, doi:10.1109/43.536723.

[48] Browne, M. C.; Clarke, E. M.; Dill, D. L.; et al. Automatic Verification of Sequential Circuits Using Temporal Logic. *IEEE Transactions on Computers*, volume C-35, no. 12, Dec 1986: pp. 1035–1044, ISSN 0018-9340, doi:10.1109/TC.1986.1676711.

[49] Coudert, O.; Madre, J. C.; Fraisse, H. A New Viewpoint on Two-Level Logic Minimization. In *Proceedings of the 30th Conference on Design Automation*, June 1993, ISSN 0738-100X, pp. 625–630, doi:10.1109/DAC.1993.204023.

[50] Fey, G.; Drechsler, R. Utilizing BDDs for disjoint SOP minimization. In *Proceedings of the 45th Midwest Symposium on Circuits and Systems*, volume 2, Aug 2002, pp. II–306–II–309 vol.2, doi:10.1109/MWSCAS.2002.1186859.

[51] Karplus, K. Using If-then-else DAGs for Multi-level Logic Minimization. In *Proceedings of the Decennial Caltech Conference on VLSI on Advanced Research in VLSI*, Cambridge, MA, USA: MIT Press, 1989, ISBN 0-262-19282-9, pp. 101–117. Available from: `http://dl.acm.org/citation.cfm?id=90897.90926`

[52] Lai, Y.-T.; Pedram, M.; Vrudhula, S. B. K. BDD Based Decomposition of Logic Functions with Application to FPGA Synthesis. In *Proceedings of the 30th Conference on Design Automation*, June 1993, ISSN 0738-100X, pp. 642–647, doi:10.1109/DAC.1993.204026.

[53] Yang, C.; Ciesielski, M. BDS: a BDD-based logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 21, no. 7, Jul 2002: pp. 866–876, ISSN 0278-0070, doi:10.1109/TCAD.2002.1013899.

[54] Vemuri, N.; Kalla, P.; Tessier, R. BDD-based Logic Synthesis for LUT-based FPGAs. *ACM Trans. Des. Autom. Electron. Syst.*, volume 7, no. 4, Oct. 2002: pp. 501–525, ISSN 1084-4309, doi:10.1145/605440.605442. Available from: `http://doi.acm.org/10.1145/605440.605442`

[55] Sasao, T.; Matsuura, M. BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition. In *Proceedings of the 42nd Design Automation Conference*, June 2005, ISSN 0738-100X, pp. 373–378, doi:10.1109/DAC.2005.193837.

[56] Bollig, B.; Wegener, I. Improving the variable ordering of OBDDs is NP-complete. *IEEE Transactions on Computers*, volume 45, no. 9, Sep 1996: pp. 993–1002, ISSN 0018-9340, doi:10.1109/12.537122.

[57] Garey, M. R.; Johnson, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990, ISBN 0716710455.

[58] Drechsler, R.; Eggersglüß, S.; Fey, G.; et al. *Test Pattern Generation Using Boolean Proof Engines*. Springer Publishing Company, Incorporated, first edition, 2009, ISBN 9048123593, 9789048123599.

[59] Ganai, M. K.; Zhang, L.; Ashar, P.; et al. Combining strengths of circuit-based and CNF-based algorithms for a high-performance SAT solver. In *Proceedings of the 39th Design Automation Conference*, 2002, ISSN 0738-100X, pp. 747–750, doi:10.1109/DAC.2002.1012722.

[60] Safarpour, S.; Veneris, A.; Drechsler, R.; et al. Managing don't cares in Boolean satisfiability. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, volume 1, Feb 2004, ISSN 1530-1591, pp. 260–265 Vol.1, doi:10.1109/DATE.2004.1268858.

[61] Chen, H.; Marques-Silva, J. TG-Pro: A SAT-based ATPG System. *Journal on Satisfiability, Boolean Modeling and Computation*, volume 8, no. 1/2, 2012: pp. 83–88.

[62] Shi, J.; Fey, G.; Drechsler, R.; et al. PASSAT: efficient SAT-based test pattern generation for industrial circuits. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI: New Frontiers in VLSI Design (ISVLSI'05)*, May 2005, ISSN 2159-3469, pp. 212–217, doi:10.1109/ISVLSI.2005.55.

[63] Park, D. Finiteness is mu-ineffable. CS-RR-003, Department of Computer Science, July 1974, d.M.R. Park, Finiteness is Mu-Ineffable, Theoretical Computer Science 3, pp. 173-181 (1976). Available from: `http://eprints.dcs.warwick.ac.uk/1126/`

[64] Kuehlmann, A.; Paruthi, V.; Krohm, F.; et al. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 21, no. 12, Dec 2002: pp. 1377–1394, ISSN 0278-0070, doi:10.1109/TCAD.2002.804386.

[65] Groote, J. F.; Zantema, H. Resolution and Binary Decision Diagrams Cannot Simulate Each Other Polynomially. Technical report, Amsterdam, The Netherlands, The Netherlands, 2000.

[66] Drechsler, R.; Fey, G.; Kinder, S. An integrated approach for combining BDD and SAT provers. In *Proceedings of the 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design (VLSID'06)*, Jan 2006, ISSN 1063-9667, pp. 6 pp.–, doi:10.1109/VLSID.2006.44.

[67] Fey, G.; Dreschler, R. Finding good counter-examples to aid design verification. In *Proceedings of the 1st ACM and IEEE International Conference on Formal Methods and Models for Co-Design*, June 2003, pp. 51–52, doi:10.1109/MEMCOD.2003.1210088.

[68] Haubelt, C.; Feldmann, R. SAT-based techniques in system synthesis. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2003, ISSN 1530-1591, pp. 1168–1169, doi:10.1109/DATE.2003.1253784.

[69] Liu, X. Improving generation method for test pattern based on BDD learning. In *Proceedings of the 10th International Conference on Electronic Measurement Instruments*, volume 3, Aug 2011, pp. 200–203, doi:10.1109/ICEMI.2011.6037887.

[70] Gaede, R. K.; Mercer, M. R.; Butler, K.; et al. CATAPULT: concurrent automatic testing allowing parallelization and using limited topology. In *Proceedings of the 25th*

ACM/IEEE Design Automation Conference, June 1988, ISSN 0738-100X, pp. 597–600, doi:10.1109/DAC.1988.14823.

[71] Srinivasan, S.; Swaminathan, G.; Aylor, J. H.; et al. Algebraic ATPG of combinational circuits using binary decision diagrams. In *Proceedings of the 3rd European Test Conference*, Apr 1993, pp. 240–248, doi:10.1109/ETC.1993.246558.

[72] Eggergluss, S.; Tille, D.; Drechsler, R. Speeding up SAT-Based ATPG Using Dynamic Clause Activation. In *Proceedings of the Asian Test Symposium*, Nov 2009, ISSN 1081-7735, pp. 177–182, doi:10.1109/ATS.2009.26.

[73] Silva, J. O. M.; Sakallah, K. A. Robust search algorithms for test pattern generation. In *Proceedings of the 27th Annual International Symposium on Fault-Tolerant Computing*, June 1997, ISSN 0731-3071, pp. 152–161, doi:10.1109/FTCS.1997.614088.

[74] Tille, D.; Eggersglüß, S.; Drechsler, R. Incremental Solving Techniques for SAT-based ATPG. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 29, no. 7, July 2010: pp. 1125–1130, ISSN 0278-0070, doi: 10.1109/TCAD.2010.2044673.

[75] Eggersglüß, S.; Drechsler, R. Efficient Data Structures and Methodologies for SAT-Based ATPG Providing High Fault Coverage in Industrial Application. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 30, no. 9, Sept 2011: pp. 1411–1415, ISSN 0278-0070, doi:10.1109/TCAD.2011.2152450.

[76] Ansotegui, C.; Manya, F. Mapping many-valued CNF formulas to Boolean CNF formulas. In *Proceedings of the 35th International Symposium on Multiple-Valued Logic (ISMVL'05)*, May 2005, ISSN 0195-623X, pp. 290–295, doi:10.1109/ISMVL.2005.23.

[77] Fey, G.; Shi, J.; Drechsler, R. Efficiency of Multi-Valued Encoding in SAT-based ATPG. In *Proceedings of the 36th International Symposium on Multiple-Valued Logic (ISMVL'06)*, May 2006, ISSN 0195-623X, pp. 25–30, doi:10.1109/ISMVL.2006.19.

[78] Eggergluss, S.; Drechsler, R. Improving Test Pattern Compactness in SAT-based ATPG. In *Proceedings of the 16th Asian Test Symposium (ATS 2007)*, Oct 2007, ISSN 1081-7735, pp. 445–452, doi:10.1109/ATS.2007.14.

[79] Gizdarski, E.; Fujiwara, H. SPIRIT: a highly robust combinational test generation algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 21, no. 12, Dec 2002: pp. 1446–1458, ISSN 0278-0070, doi:10.1109/TCAD.2002.804387.

[80] Mahlstedt, U.; Gruning, T.; Ozcan, C.; et al. Contest: a fast ATPG tool for very large combinational circuits. In *Proceedings of the IEEE International Conference on Computer-Aided Design*, Nov 1990, pp. 222–225, doi:10.1109/ICCAD.1990.129886.

[81] Henftling, M.; Wittmann, H. C.; Antreich, K. J. A single-path-oriented fault-effect propagation in digital circuits considering multiple-path sensitization. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov 1995, ISSN 1092-3152, pp. 304–309, doi:10.1109/ICCAD.1995.480027.

[82] Waicukauski, J. A.; Shupe, P. A.; Giramma, D. J.; et al. ATPG for ultra-large structured designs. In *Proceedings of the International Test Conference*, Sep 1990, pp. 44–51, doi:10.1109/TEST.1990.113999.

[83] Kunz, W.; Pradhan, D. K. Recursive learning: a new implication technique for efficient solutions to CAD problems-test, verification, and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 13, no. 9, Sep 1994: pp. 1143–1158, ISSN 0278-0070, doi:10.1109/43.310903.

[84] Kirkland, T.; Mercer, M. R. A Topological Search Algorithm for ATPG. In *Proceedings of the 24th Conference on Design Automation*, June 1987, ISSN 0738-100X, pp. 502–508, doi:10.1109/DAC.1987.203289.

[85] Schulz, M. H.; Auth, E. Improved deterministic test pattern generation with applications to redundancy identification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 8, no. 7, Jul 1989: pp. 811–816, ISSN 0278-0070, doi:10.1109/43.31539.

[86] Kundu, S.; Huisman, L. M.; Nair, I.; et al. A small test generator for large designs. In *Proceedings of the International Test Conference*, Sept 1992, ISSN 1089-3539, pp. 30–40, doi:10.1109/TEST.1992.527801.

[87] Brglez, F.; Fujiwara, H. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Proc. of International Symposium on Circuits and Systems*, 1985, pp. 663–698.

[88] Brglez, F.; Bryan, D.; Kozminski, K. Combinational profiles of sequential benchmark circuits. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 1989, pp. 1929–1934 vol.3, doi:10.1109/ISCAS.1989.100747.

[89] Corno, F.; Reorda, M. S.; Squillero, G. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design Test of Computers*, volume 17, no. 3, Jul 2000: pp. 44–53, ISSN 0740-7475, doi:10.1109/54.867894.

[90] Bhattacharya, D.; Agrawal, P.; Agrawal, V. D. Delay fault test generation for scan/hold circuits using Boolean expressions. In *Proceedings of the 29th ACM/IEEE Design Automation Conference*, Jun 1992, ISSN 0738-100X, pp. 159–164, doi:10.1109/DAC.1992.227843.

[91] Fuchs, K.; Fink, F.; Schulz, M. H. DYNAMITE: an efficient automatic test pattern generation system for path delay faults. *IEEE Transactions on Computer-Aided*

Design of Integrated Circuits and Systems, volume 10, no. 10, Oct 1991: pp. 1323–1335, ISSN 0278-0070, doi:10.1109/43.88928.

[92] S.M. Reddy, C. L.; I., S. P. An automatic test pattern generator for the detection of path delay faults. In *In Int'l Conf. on CAD*, 1987, pp. 284–287.

[93] Drechsler, R. BiTeS: A BDD Based Test Pattern Generator for Strong Robust Path Delay Faults. In *Proceedings of the Conference on European Design Automation*, EURO-DAC '94, Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, ISBN 0-89791-685-9, pp. 322–327. Available from: `http://dl.acm.org/citation.cfm?id=198174.198276`

[94] Jacobi, R. P. *A Study of the Application of Binary Decision Diagrams in Multilevel Logic Synthesis*. Dissertation thesis, Universite Catholique de Louvain. Faculte de Sciences Appliquees. Laboratoire de Microelectronique, 1993.

[95] Swamy, G.; McGeer, P.; Brayton, R. K. A Fully Implicit Quine-McCluskey Procedure Using BDD's. Technical report UCB/ERL M92/127, EECS Department, University of California, Berkeley, 1992. Available from: `http://www.eecs.berkeley.edu/Pubs/TechRpts/1992/2213.html`

[96] Yang, S. Logic Synthesis and Optimization Benchmarks User Guide Version 3.0. Technical report, Microelectronics Center of North Carolina, January 1991.

[97] Coudert, O. Doing Two-Level Logic Minimization 100 Times Faster. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, 1995, pp. 112–121. Available from: `http://dl.acm.org/citation.cfm?id=313651.313674`

[98] Sapra, S.; Theobald, M.; Clarke, E. SAT-based algorithms for logic minimization. In *Proceedings of the 21st International Conference on Computer Design*, Oct 2003, ISSN 1063-6404, pp. 510–517, doi:10.1109/ICCD.2003.1240948.

[99] Rudell, R. L.; Sangiovanni-Vincentelli, A. Multiple-Valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 6, no. 5, September 1987: pp. 727–750, ISSN 0278-0070, doi:10.1109/TCAD.1987.1270318.

[100] Pizzuti, C. Computing prime implicants by integer programming. In *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence*, Nov 1996, ISSN 1082-3409, pp. 332–336, doi:10.1109/TAI.1996.560473.

[101] Wittgenstein, H. P. Tractatus Logico-Philosophicus. *Routledge and Kegan Paul*, 1961.

[102] Dantzig, G. *Linear programming and extensions.* Rand Corporation Research Study, Princeton, NJ: Princeton Univ. Press, 1963. Available from: `http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+180926950&sourceid=fbw_bibsonomy`

[103] Manquinho, V. M.; Flores, P. F.; Silva, J. P. M.; et al. Prime implicant computation using satisfiability algorithms. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence*, Nov 1997, ISSN 1082-3409, pp. 232–239, doi:10.1109/TAI.1997.632261.

[104] Fey, G.; Drechsler, R. Minimizing the number of paths in BDDs. In *Proceedings of the 15th Symposium on Integrated Circuits and Systems Design*, 2002, pp. 359–364, doi:10.1109/SBCCI.2002.1137683.

[105] Rudell, R. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov 1993, pp. 42–47, doi:10.1109/ICCAD.1993.580029.

[106] Berman, C. L. Circuit Width, Register Allocation, and Reduced Function Graphs. Technical report Technical Report RC14129, IBM Thomas J. Watson Research Center, January 1988.

[107] Friedman, S. J.; Supowit, K. J. Finding the Optimal Variable Ordering for Binary Decision Diagrams. In *Proceedings of the 24th Conference on Design Automation*, June 1987, ISSN 0738-100X, pp. 348–356, doi:10.1109/DAC.1987.203267.

[108] Ashenhurst, R. The decomposition of switching functions. In *Int. Symp. on the Theory of Switching*, volume 29, 1959, pp. 74–116.

[109] Curtis, H. *A new approach to the design of switching circuits*. D. Van Nostrand Company (1962), 1962, ISBN 0442017944, 9780442017941.

[110] Roth, J. P.; Karp, R. M. Minimization Over Boolean Graphs. *IBM Journal of Research and Development*, volume 6, no. 2, April 1962: pp. 227–238, ISSN 0018-8646, doi:10.1147/rd.62.0227.

[111] Selman, B. Stochastic Search and Phase Transitions: AI Meets Physics. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, ISBN 1-55860-363-8, 978-1-558-60363-9, pp. 998–1002. Available from: `http://dl.acm.org/citation.cfm?id=1625855.1625985`

[112] Davis, M.; Putnam, H. A Computing Procedure for Quantification Theory. *J. ACM*, volume 7, no. 3, jul 1960: pp. 201–215, ISSN 0004-5411, doi:10.1145/321033.321034. Available from: `http://doi.acm.org/10.1145/321033.321034`

[113] Kravchuk, O.; Pullan, W.; Thornton, J.; et al. An Investigation of Variable Relationships in 3-SAT Problems. In *Proceedings of the 15th Australian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, AI '02, London, UK, UK: Springer-Verlag, 2002, ISBN 3-540-00197-2, pp. 579–590. Available from: `http://dl.acm.org/citation.cfm?id=646079.678569`

[114] H. Zhang, M. S. An efficient algorithm for unit propagation. In *Proceedings of the 4th International Symposium on Artificial Intelligence and Mathematics*, 1996.

[115] Zorian, Y.; Marinissen, E. J.; Dey, S. Testing embedded-core-based system chips. *Computer*, volume 32, no. 6, Jun 1999: pp. 52–60, ISSN 0018-9162, doi:10.1109/2.769444.

[116] E.J. Marinissen, M. L. T. M. M. R. Y. Z., R. Kapur. On IEEE P1500's Standard for Embedded Core Test. *J. Electronic Testing*, volume 18, no. 4-5, 2002: pp. 365–383, doi:10.1023/A:1016585206097. Available from: `http://dx.doi.org/10.1023/A:1016585206097`

[117] Krishna, C. V.; Touba, N. A. Reducing test data volume using LFSR reseeding with seed compression. In *Proceedings of the International Test Conference*, 2002, ISSN 1089-3539, pp. 321–330.

[118] Gonciari, P. T.; Al-Hashimi, B. M.; Nicolici, N. Variable-length input Huffman coding for system-on-a-chip test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 22, no. 6, June 2003: pp. 783–796, ISSN 0278-0070, doi:10.1109/TCAD.2003.811451.

[119] Ichihara, H.; Ogawa, A.; Inoue, T.; et al. Dynamic test compression using statistical coding. In *Proceedings of the 10th Asian Test Symposium*, 2001, ISSN 1081-7735, pp. 143–148, doi:10.1109/ATS.2001.990273.

[120] Ichihara, H.; Kinoshita, K.; Pomeranz, I.; et al. Test transformation to improve compaction by statistical encoding. In *Proceedings of the 13th International Conference on VLSI Design*, 2000, ISSN 1063-9667, pp. 294–299, doi:10.1109/ICVD.2000.812624.

[121] Jas, A.; Ghosh-Dastidar, J.; Touba, N. A. Scan vector compression/decompression using statistical coding. In *Proceedings of the 17th IEEE VLSI Test Symposium*, 1999, ISSN 1093-0167, pp. 114–120, doi:10.1109/VTEST.1999.766654.

[122] Chandra, A.; Chakrabarty, K. Frequency-directed run-length (FDR) codes with application to system-on-a-chip test data compression. In *Proceedings of the 19th IEEE VLSI Test Symposium*, 2001, pp. 42–47, doi:10.1109/VTS.2001.923416.

[123] Jas, A.; Touba, N. A. Test vector decompression via cyclical scan chains and its application to testing core-based designs. In *Proceedings of the International Test Conference*, Oct 1998, ISSN 1089-3539, pp. 458–464, doi:10.1109/TEST.1998.743186.

[124] Chandra, A.; Chakrabarty, K. Test data compression for system-on-a-chip using Golomb codes. In *Proceedings of the 18th IEEE VLSI Test Symposium*, 2000, ISSN 1093-0167, pp. 113–120, doi:10.1109/VTEST.2000.843834.

[125] Bayraktaroglu, I.; Orailoglu, A. Test volume and application time reduction through scan chain concealment. In *Proceedings of the Design Automation Conference*, 2001, ISSN 0738-100X, pp. 151–155, doi:10.1109/DAC.2001.156125.

[126] Reddy, S. M.; Miyase, K.; Kajihara, S.; et al. On test data volume reduction for multiple scan chain designs. In *Proceedings of the 20th IEEE VLSI Test Symposium*, 2002, pp. 103–108, doi:10.1109/VTS.2002.1011119.

[127] Das, D.; Touba, N. A. Reducing test data volume using external/LBIST hybrid test patterns. In *Proceedings of the International Test Conference*, 2000, ISSN 1089-3539, pp. 115–122, doi:10.1109/TEST.2000.894198.

[128] Rajski, J.; Tyszer, J.; Kassab, M.; et al. Embedded deterministic test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 23, no. 5, May 2004: pp. 776–792, ISSN 0278-0070, doi:10.1109/TCAD.2004.826558.

[129] Eén, N.; Sörensson, N. An Extensible SAT-solver. *Theory and Applications of Satisfiability Testing*, volume 2919, 2004: pp. 333–336, doi:10.1007/978-3-540-24605-3_37. Available from: `http://dx.doi.org/10.1007/978-3-540-24605-3_37`

[130] Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; et al. Chaff: engineering an efficient SAT solver. In *Proceedings of the 39th Design Automation Conference*, 2001, ISSN 0738-100X, pp. 530–535, doi:10.1109/DAC.2001.156196.

[131] Moskewicz, M. W.; Madigan, C. F.; Zhao, Y.; et al. Chaff: engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, 2001, ISSN 0738-100X, pp. 530–535, doi:10.1109/DAC.2001.156196.

[132] Ben-Eliyahu, R.; Dechter, R. On Computing Minimal Models. In *Proceedings of the 11th National Conference on Artificial Intelligence*, 1993, pp. 2–8. Available from: `http://www.aaai.org/Library/AAAI/1993/aaai93-001.php`

[133] Ravi, K.; Somenzi, F. Minimal Assignments for Bounded Model Checking. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2004, pp. 31–45, doi:10.1007/978-3-540-24730-2_3. Available from: `http://dx.doi.org/10.1007/978-3-540-24730-2_3`

[134] Dillig, I.; Dillig, T.; McMillan, K. L.; et al. In *Proceedings of the 24th International Conference on Computer Aided Verification*, Springer International Publishing, pp. 394–409, doi:10.1007/978-3-642-31424-7_30.

[135] Boros, E.; Hammer, P. L. Pseudo-boolean Optimization. *Discrete Appl. Math.*, volume 123, no. 1-3, nov 2002: pp. 155–225, ISSN 0166-218X, doi:10.1016/S0166-218X(01)00341-9. Available from: `http://dx.doi.org/10.1016/S0166-218X(01)00341-9`

[136] Fummi, F.; Sciuto, D. Implicit test pattern generation constrained to cellular automata embedding. In *Proceedings of the 15th IEEE VLSI Test Symposium*, Apr 1997, ISSN 1093-0167, pp. 54–59, doi:10.1109/VTEST.1997.599441.

[137] Liu, X.; Hsiao, M. S. Constrained ATPG for broadside transition testing. In *Proceedings of the 18th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Nov 2003, ISSN 1550-5774, pp. 175–182, doi:10.1109/DFTVS.2003.1250110.

[138] Konijnenburg, M.; van der Linden, J.; van de Goor, A. Automatic test pattern generation for industrial circuits with restrictors. *Microelectronics Journal*, volume 26, no. 7, 1995: pp. 635 – 645, ISSN 0026-2692, doi:http://dx.doi.org/10.1016/0026-2692(95)00005-3. Available from: `http://www.sciencedirect.com/science/article/pii/0026269295000053`

[139] Dorsch, R.; Wunderlich, H.-J. Reusing Scan Chains for Test Pattern Decompression. In *Proceedings of the 6th European Test Workshop (ETW)*, Institute of Electrical and Electronics Engineers, Mai 2001, ISBN 0-7695-10 16-7, ISSN 1530-1877, pp. 124–132. Available from: `http://www2.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=INPROC-2001-75&engl=0`

[140] Dorsch, R.; Wunderlich, H. Reusing Scan Chains for Test Pattern Decompression. *J. Electronic Testing*, volume 18, no. 2, 2002: pp. 231–240, doi:10.1023/A:1014968930415. Available from: `http://dx.doi.org/10.1023/A:1014968930415`

[141] Daehn, W.; Mucha, J. Hardware Test Pattern Generation for Built-In Testing. In *Proceedings of the International Test Conference*, 1981, pp. 110–120.

[142] Ayari, B.; Kaminska, B. A new dynamic test vector compaction for automatic test pattern generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 13, no. 3, Mar 1994: pp. 353–358, ISSN 0278-0070, doi:10.1109/43.265676.

[143] Balaz, M.; Dobai, R.; Gramatova, E. Delay faults testing. In *Proceedings of the Design and Test Technology for Dependable Systems-on-Chip*, Eds. Information Science Publishing, 2011, doi:10.4018/978-1-60960-212-3.

[144] Dobai, R.; Balaz, M. SAT-Based Generation of Compressed Skewed-Load Tests for Transition Delay Faults. In *Proceedings of the 14th Euromicro Conference on Digital System Design*, Aug 2011, pp. 191–196, doi:10.1109/DSD.2011.28.

[145] Tille, D.; Eggersglüß, S.; Le, H. M.; et al. Structural heuristics for SAT-based ATPG. In *Proceedings of the 17th IFIP International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2009, ISSN 2324-8432, pp. 77–82, doi:10.1109/VLSISOC.2009.6041334.

[146] Eggersglüß, S.; Drechsler, R. Robust algorithms for high quality Test Pattern Generation using Boolean Satisfiability. In *Proceedings of the IEEE International Test Conference*, Nov 2010, ISSN 1089-3539, pp. 1–10, doi:10.1109/TEST.2010.5699289.

[147] Lee, H.; Ha, D. Atalanta: an Efficient ATPG for Combinational Circuits. Technical report Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1999.

[148] Jeníček, J.; Novák, O. COMPAS - Advanced test compressor. In *Proceedings of the IEEE East-West Design and Test Symposium*, Sept 2010, pp. 543–548, doi:10.1109/EWDTS.2010.5742114.

[149] Jenicek, J.; Novak, O. Test Pattern Compression Based on Pattern Overlapping. In *Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems*, April 2007, pp. 1–6, doi:10.1109/DDECS.2007.4295250.

[150] Baik, D. H.; Saluja, K. K.; Kajihara, S. Random access scan: a solution to test power, test data volume and test time. In *Proceedings of the 17th International Conference on VLSI Design*, 2004, pp. 883–888, doi:10.1109/ICVD.2004.1261042.

[151] Baik, D. H.; Saluja, K. K. Progressive random access scan: a simultaneous solution to test power, test data volume and test time. In *Proceedings of the IEEE International Conference on Test*, Nov 2005, ISSN 1089-3539, pp. 10 pp.–368, doi:10.1109/TEST.2005.1583994.

[152] Hamzaoglu, I.; Patel, J. H. Reducing test application time for full scan embedded cores. In *Proceedings of the 29th Annual International Symposium on Fault-Tolerant Computing*, June 1999, ISSN 0731-3071, pp. 260–267, doi:10.1109/FTCS.1999.781060.

[153] Hamzaoglu, I.; Patel, J. H. Test set compaction algorithms for combinational circuits. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov 1998, pp. 283–289, doi:10.1109/ICCAD.1998.144279.

[154] Chandra, A.; Chakrabarty, K. Test data compression and test resource partitioning for system-on-a-chip using frequency-directed run-length (FDR) codes. *IEEE Transactions on Computers*, volume 52, no. 8, Aug 2003: pp. 1076–1088, ISSN 0018-9340, doi:10.1109/TC.2003.1223641.

[155] Puggelli, A.; Welp, T.; Kuehlmann, A.; et al. Are logic synthesis tools robust? In *Proceedings of the 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2011, ISSN 0738-100x, pp. 633–638.

[156] Eén, N.; Sörensson, N. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, volume 2, 2006: pp. 1–26.

[157] Lee, H. K.; Ha, D. S. HOPE: an efficient parallel fault simulator for synchronous sequential circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 15, no. 9, Sep 1996: pp. 1048–1058, ISSN 0278-0070, doi:10.1109/43.536711.

[158] McElvain, K. LGSynth93 Benchmark Set: Version 4.0. may 1993.

[159] Singer, J.; Gent, I. P.; Smaill, A. Local Search on Random 2+p-SAT. In *Proceedings of the 14th European Conference on Artificial Intelligence*, 2000, pp. 113–117.

[160] Zhang, L.; Madigan, C. F.; Moskewicz, M. H.; et al. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, Nov 2001, ISSN 1092-3152, pp. 279–285, doi:10.1109/ICCAD.2001.968634.

[161] E.Gizdarski; H.Fujiwara. A New Data Structure for Complete Implication Graph with Application for Static Learning. Technical report Technical report NAIST-IS-TR2000-001, January 2000.

[162] Tafertshofer, P.; Ganz, A. SAT based ATPG using fast justification and propagation in the implication graph. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, Nov 1999, ISSN 1092-3152, pp. 139–146, doi:10.1109/ICCAD.1999.810638.

[163] Mitra, S.; McCluskey, E. J. Which concurrent error detection scheme to choose ? In *Proceedings of the International Test Conference*, 2000, ISSN 1089-3539, pp. 985–994, doi:10.1109/TEST.2000.894311.

[164] Pradhan, D. K. *Fault-tolerant Computer System Design*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996, ISBN 0-13-057887-8.

[165] Czutro, A.; Polian, I.; Lewis, M.; et al. TIGUAN: Thread-Parallel Integrated Test Pattern Generator Utilizing Satisfiability ANalysis. In *Proceedings of the 22nd International Conference on VLSI Design*, Jan 2009, ISSN 1063-9667, pp. 227–232, doi:10.1109/VLSI.Design.2009.20.

[166] Hunger, M.; Hellebrand, S. Verification and Analysis of Self-Checking Properties through ATPG. In *Proceedings of the 14th IEEE International On-Line Testing Symposium*, July 2008, ISSN 1942-9398, pp. 25–30, doi:10.1109/IOLTS.2008.32.

[167] Sawhney, P.; Ganesh, G.; Bhattacharjee, A. K. Automatic Construction of Runtime Monitors for FPGA Based Designs. In *Proceedings of the International Symposium on Electronic System Design*, Dec 2011, pp. 164–169, doi:10.1109/ISED.2011.70.

[168] Prasad, M. R.; Chong, P.; Keutzer, K. Why is ATPG Easy? In *Proceedings of the 36th Annual ACM/IEEE Design Automation Conference*, New York, NY, USA:

ACM, 1999, ISBN 1-58113-109-7, pp. 22–28, doi:10.1145/309847.309857. Available from: http://doi.acm.org/10.1145/309847.309857

[169] Mitra, S.; McCluskey, E. J. Diversity techniques for concurrent error detection. In *Proceedings of the International Symposium on Quality Electronic Design*, 2001, pp. 249–250, doi:10.1109/ISQED.2001.915234.

[170] L. Kafka, H. K., P. Kubalik. Fault classification for self-checking circuits implemented in FPGA. In *Proceedings of the 8th IEEE Workshop on Design and Diagnostics of Electronics Circuits and Systems*, 2005, pp. 228–231.

[171] Micheli, G. D. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, first edition, 1994, ISBN 0070163332.

[172] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. Available from: http://www.eecs.berkeley.edu/~alanmi/abc/

[173] Ubar, R.; Jrimgi, L.; Raik, J. Shared Structurally Synthesized BDDs for speeding-up parallel pattern simulation in digital circuits. In *Proceedings of the Nordic Circuits and Systems Conference (NORCAS): NORCHIP International Symposium on System-on-Chip (SoC)*, Oct 2015, pp. 1–4, doi:10.1109/NORCHIP.2015.7364406.

[174] Ubar, R. *Test generation for digital systems based on alternative graphs*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, ISBN 978-3-540-48785-2, pp. 149–164, doi:10.1007/3-540-58426-9_129. Available from: http://dx.doi.org/10.1007/3-540-58426-9_129

[175] M. Kubica, D. K. SMTBDD : New Form of BDD for Logic Synthesis. *International Journal of Electronics and Telecommunications*, volume 62, no. 1, April 2016: pp. 33–41, ISSN 2300-1933, doi:10.1515/eletel-2016-0004.

[176] Ubar, R.; Mironov, D.; Raik, J.; et al. Structurally synthesized multiple input BDDs for simulation of digital circuits. In *Proceedings of the 16th IEEE International Conference on Electronics, Circuits, and Systems*, Dec 2009, pp. 451–454, doi:10.1109/ICECS.2009.5410895.

Reviewed Publications of the Author Relevant to the Thesis

[A.1] Balcárek, J.; Fišer, P.; Schmidt, J. *On don't cares in test compression.* Microprocessors and Microsystems, Volume 38, Issue 8, pp. 754-765, Nederland, Amsterdam, 2014.

[A.2] Balcárek, J.; Fišer, P.; Schmidt, J. *PBO-Based Test Compression.* 17th Euromicro Conference on Digital Systems Design, Italy, Verona, pp. 679-682, 2014.

[A.3] Fišer, P.; Schmidt, J.; Balcárek, J. *Sources of Bias in EDA Tools and Its Influence.* 17th IEEE International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS), Poland, Warsaw, pp. 258-261, 2014.

[A.4] Schmidt, J.; Fišer, P.; Balcárek, J. *On Robustness of EDA Tools.* 17th Euromicro Conference on Digital Systems Design, pp. 427-434, Italy, Verona, 2014.

[A.5] Balcárek, J.; Fišer, P.; Schmidt, J. *Techniques for SAT-Based Constrained Test Pattern Generation.* Microprocessors and Microsystems, Volume 37, Issue 2, pp. 185-195, Nederland, Amsterdam, 2013.

The paper has been cited in:

- Eggersgluess, S.; Wille, R.; Drechsler, R. *Improved SAT-based ATPG: More Constraints, Better Compaction*, ICCAD-IEEE ACM International Conference on Computer-Aided Design, pp. 85-90, 2013.

[A.6] Schmidt, J.; Fišer, P.; Balcárek, J. *The Influence of Implementation Type on Dependability Parameters.* Microprocessors and Microsystems, Volume 37, Issues 6-7, pp. 641-648, Nederland, Amsterdam, 2013.

[A.7] Balcárek, J.; Fišer, P.; Schmidt, J. *Simulation and SAT Based ATPG for Compressed Test Generation.* 16th Euromicro Conference on Digital Systems Design, España, Piscataway, pp. 445-452, 2013.

[A.8] Schmidt, J.; Fišer, P.; Balcárek, J. *The Influence of Implementation Technology on Dependability Parameters.* 15th Euromicro Conference on Digital Systems Design, pp. 368-373, Turkey, Izmir, 2012.

[A.9] Balcárek, J.; Fišer, P.; Schmidt, J. *Techniques for SAT-Based Constrained Test Pattern Generation.* 14th Euromicro Conference on Digital System Design, pp. 360-366, Finland, Oulu, 2011.

The paper has been cited in:

- Dobai, R.; Balaz, M. *Genetic Method for Compressed Skewed-Load Delay Test Generation*, IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 242-247, 2012.
- Dobai, R.; Balaz, M. *SAT-based generation of compressed skewed-load tests for transition delay faults*, Microprocessors and Microsystems, pp. 196-205, 2013.
- Yamashita, J.; Yotsuyanagi, H.; Hashizume, M.; Kinoshita, K. *SAT-Based Test Generation for Open Faults Using Fault Excitation Caused by Effect of Adjacent Lines*, IEICE Transactions on Fundamentals of Electronics Communications and Computer Science, pp. 2561-2567, 2013.
- Dobai, R.; Balaz, M. *Compressed Skewed-load Delay Test Generation Based on Evolution and Deterministic Initialization of Populations*, Computiong and Informatics, pp. 251-272, 2013.
- Novák, O.; Jeníček, J.; Rozkovec, M. *LFSR Reseeding Based Test Compression Respecting Different Controllability of Decompressor Outputs*, Design and Diagnostics of Electronic Circuits & Systems (DDECS), pp. 9-14, 2015.

[A.10] Balcárek, J.; Fišer, P.; Schmidt, J. *Test Patterns Compression Technique Based on a Dedicated SAT-Based ATPG.* 13th Euromicro Conference on Digital Systems Design, France, Lille, pp. 805-808, 2010.

The paper has been cited in:

- Dobai, R.; Balaz, M. *Genetic Method for Compressed Skewed-Load Delay Test Generation*, IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 242-247, 2012.
- Dobai, R.; Balaz, M. *SAT-based generation of compressed skewed-load tests for transition delay faults*, Microprocessors and Microsystems, pp. 196-205, 2013.
- Dobai, R.; Balaz, M. *Compressed Skewed-load Delay Test Generation Based on Evolution and Deterministic Initialization of Populations*, Computiong and Informatics, pp. 251-272, 2013.
- Chloupek, M.; Novák, O. *Test Pattern Compression Based on Pattern Overlapping and Broadcasting*, International Workshop on Electronics Control Measurement and Signals, pp. 56-60, 2011.

- ◦ Chloupek, M.; Novák, O. *On Test Time Reduction Using Pattern Overlapping, Broadcasting and On-Chip Decompression*, IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems, pp. 300-305, 2012.
- ◦ Dobai, R.; Balaz, M. *SAT-Based Generation of Compressed Skewed-Load Tests for Transition Delay Faults*, 14th Euromicro Conference on Digital Systems Design, pp. 191-196, 2011.
- ◦ Chloupek, M.; Novák, O. *Scan chain configuration method for broadcast decompressor architecture*, 12th Latin American Test Workshop (LATW), p. 5, 2011.
- ◦ Chloupek, M.; Novák, O. *Test pattern compression based on pattern overlapping and broadcasting*, 10th International Workshop on Electronics, Control, Measurement and Signals (ECMS), p. 5, 2011.
- ◦ L. Cvetković, D. Dražić. *Fault Detection in Combinational Circuits Using Boolean Satisfiability*, International Journal of Computer Science and Information Technology and Security (IJCSITS), pp. 804-808, 2012.

[A.11] Balcárek, J.; Fišer, P.; Schmidt, J. *On Properties of SAT Instances Produced by SAT-Based ATPGs.* 5th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, Czech Republic, Brno, pp. 3-10, 2009.

Remaining Publications of the Author Relevant to the Thesis

[A.12] Balcárek, J.; Fišer, P.; Schmidt, J. *Generalized Miter and its Application in Hardware Design.* 8th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS), Czech Republic, Znojmo, pp. 119-120, 2012.

[A.13] Balcárek, J.; Fišer, P.; Schmidt, J. *Implicit Techniques for Constrained Test Patterns Generation.* 7th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, Czech Republic, Brno, p. 106, 2011.

[A.14] Balcárek, J. *Implicit Representations in the Diagnostics of The Digital Circuits.* Počítačové architektury & diagnostika, pp. 67-72, Slovakia, Stará Lesná, 2011.

[A.15] Balcárek, J.; Fišer, P.; Schmidt, J. *Implicit Representations in Test Patterns Compression for Scan-Based Digital Circuits.* 15th IEEE European Test Symposium, Czech Republic, Prague, 2010.

[A.16] Balcárek, J. *Implicit Rrepresentations in Customized Testing of Digital Circuits.* Počítačové architektury & diagnostika, pp. 15-20, Czech Republic, Brno, 2010.

[A.17] Balcárek, J. *Test Patterns Compression Technique Based on SAT Solving for Scan-Based Digital Circuits.* Počítačové architektury & diagnostika, pp. 26-31, Czech Republic, Zlín, 2009.