Institute of Computer Architecture and Computer Engineering
Prof. Dr. rer. nat. habil. Hans-Joachim Wunderlich
Pfaffenwaldring 47, 70569 Stuttgart

Master Thesis Nr. 3447

Embedding Deterministic Patterns in Partial Pseudo-Exhaustive Test

Anastasia Sannikova

MSc Thesis

in partial fulfillment of the requirements

for the degree of **Master of Science**

Supervisors: M. Sc. Abdullah Mumtaz

Dipl.–Inf. Michael Imhof

Examiner: Prof. Dr. rer. nat. habil. Hans-Joachim Wunderlich

Start Date: 15.11.2012

Submission Date: 17.05.2013

CR Classification: B.7.3, B.8.1

Study Program: M. Sc. Information Technology (INFOTECH)

To my beloved husband

ABSTRACT

The topic of this thesis is related to testing of very large scale integration circuits. The thesis presents the idea of optimizing mixed-mode built-in self-test (BIST) scheme. Mixed-mode BIST consists of two phases. The first phase is pseudo-random testing or partial pseudo-exhaustive testing (P-PET). For the faults not detected by the first phase, deterministic test patterns are generated and applied in the second phase. Hence, the defect coverage of the first phase influences the number of patterns to be generated and stored. The advantages of P-PET in comparison with usual pseudo-random test are in obtaining higher fault coverage and reducing the number of deterministic patterns in the second phase of mixed-mode BIST. Test pattern generation for P-PET is achieved by selecting characteristic polynomials of multiple-polynomial linear feedback shift register (MP-LFSR).

In this thesis, the mixed-mode BIST scheme with P-PET in the first phase is further improved in terms of the fault coverage of the first phase. This is achieved by optimization of polynomial selection of P-PET.

In usual mixed-mode BIST, the set of undetected by the first phase faults is handled in the second phase by generating deterministic test patterns for them. The method in the thesis is based on consideration of these patterns during polynomial selection. In other words, we are embedding deterministic test patterns in P-PET. In order to solve the problem, the algorithm for the selection of characteristic polynomials covering the pre-generated patterns is developed.

The advantages of the proposed approach in terms of the defect coverage and the number of faults left after the first phase are presented using contemporary industrial circuits. A comparison with usual pseudo-random testing is also performed. The results prove the benefits of P-PET with embedded test patterns in terms of the fault coverage, while maintaining comparable test length and time.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
CUT	Circuit under Test
DFT	Design for Testability
ET	Exhaustive Testing
LFSR	Linear Feedback Shift Register
MISR	Multiple Input Signature Register
MP-LFSR	Multiple-Polynomial LFSR
PET	Pseudo-Exhaustive Testing
PI	Primary Input
PO	Primary Output
P-PET	Partial Pseudo-Exhaustive Testing
PPI	Pseudo-Primary Input
PPO	Pseudo-Primary Output
PRT	Pseudo-Random Testing
ROM	Read-Only Memory
PRSG	Parallel Shift Register Sequence Generator
RA	Response Analyzer
SAT	Satisfiability
STUMPS	Self-Test Using Multiple Input Signature Register and Parallel Shift Register Sequence Generator
TPG	Test Pattern Generator
VLSI	Very Large Scale Integration

CHAPTER 1

INTRODUCTION

Contents

1.1 Motivation and goals of the work

Testing plays a crucial role in any kind of production. There is a certain risk of the product to be of low-quality when poorly examined. The meaning of the term testing varies depending on the device to be tested. In general sense, testing is a checking of the quality of the item. The results of the testing show whether the product meets all specifications and requirements.

For very large scale integration (VLSI) circuits, testing is a necessity due to possible defect presence. These defects might be caused by material defects, malfunctions of equipment, design errors. On the stage of development, when the design is fabricated for the first time, testing is performed to ensure correctness of the design and meeting of specifications. In mass production, every chip is subjected to manufacturing testing where material defects are of concern. In testing, the representation of the material defect, fault, is used. There exist different fault models to capture the nature of the physical defect logically. The most spread fault model is the stuck-at fault model. In this model, it is assumed that the line is always connected to ground (stuck-at “0”) or power supply (stuck-at “1”).

VLSI circuits are tested by applying the test patterns to the inputs of the circuit and comparing each of the output to the correct one. The test inputs, or patterns, need to be pre-computed and stored. They also can be generated during the test. Testing of VLSI circuits can be classified to online testing, which takes place during system operation, and offline; external testing and internal, or self-testing. External testing is performed by automatic test equipment (ATE). For the purpose of pattern generation, automatic test pattern generator (ATPG) is used. Internal testing takes advantages of design for testability (DFT) techniques to decrease the cost and the time of testing. DFT is a technique which increases the testability of the design. One well-known application of DFT is built-in self-test (BIST). In BIST, the part of the circuit on a chip tests the circuitry by generating the test patterns and analyzing the output responses of the circuit.

Testing of VLSI circuits can be fault model dependent and fault model independent. Generally speaking, the fault model dependent algorithm creates test for the faults of a pre-defined fault model. In the fault model independent testing, test patterns are generated without targeting specific fault models. The fault model dependent testing has a smaller set of the patterns to be applied, but it is more difficult to generate these patterns; whereas the fault model independent test set is easier to generate, but it ends up with larger number of the patterns to ensure specified fault coverage. Even so a fault model is helpful in the testing; it cannot completely accurately represent the real defect [18]. So, the significant advantage of the fault model independent testing is that it is not constrained to the specific fault model.

As an example of the fault model dependent testing, pseudo-random testing can be named. In *pseudo-random* testing [10], the set of pseudo-random patterns are used to test the circuit. If we have a circuit with n inputs, then the subset of 2^n test patterns is applied. Pseudo-random testing cannot give 100% fault coverage since the circuit contains random-pattern-resistant faults. In order to determine the quality of the test, a fault simulation is required. In addition to that, the number of patterns needed to ensure high fault coverage might be large. This makes the approach impractical.

In order to increase the fault coverage of pseudo-random testing, it is often used as a part of *mixed-mode BIST* scheme. Mixed-mode BIST consists of two phases. The first phase is usually pseudo-random testing covering the most of the faults, and the second is the phase of generation of deterministic test patterns to cover all faults undetected in the first phase, so-called random-pattern resistant faults. The limitations of the fault model to represent a real defect lead us to the consideration of fault-model independent testing.

We will start by introduction of e*xhaustive testing* (ET). Exhaustive testing gives 100% defect coverage, but it requires applying all possible 2^n test patterns to an n-input combinational circuit under test (CUT) and verifying the correctness of the output for each combination. If n is large, exhaustive testing requires too much time and certainly very large number of the test patterns to be applied. This makes the approach unrealizable.

A way to decrease the test time is *pseudo-exhaustive testing* (PET). It uses 2^w test patterns, where $w<n$. Pseudo-exhaustive testing [11] exercises different kinds of circuit segmentation, and then it tests each segment exhaustively. The time for PET depends on the sizes of segments. The obvious advantage of PET is that it does not require fault simulation or fault modeling [15].

In ET and PET one can obtain 100% defect coverage. However, the sizes of the circuits, as well as the number of primary inputs and outputs are increasing. Mentioned approaches are facing the challenge of generating long test length for today's circuits, and that makes them infeasible. In order to solve this problem *partial pseudo-exhaustive testing* (P-PET) was proposed.

P-PET [9] exercises cone segmentation where each cone is a subcircuit containing all predecessors of the output of the circuit. In P-PET case, only cones up to a given size *MAX* are tested exhaustively (instead of all the circuit cones as in pseudo-exhaustive testing and instead of exhaustively enumerating of all inputs as in exhaustive testing). After applying P-PET, some faults from the larger than *MAX* cones are left undetected and the desired fault coverage might not be achieved. In order to increase the fault coverage, mixed-mode BIST scheme is needed.

In [9] it was proposed to use P-PET in the first phase of mixed-mode BIST. The benefit of the approach is in obtaining higher fault coverage than in pseudo-random testing while maintaining the handleable test length. After applying P-PET, fewer faults are left for the second phase of mixed-mode BIST in comparison with usage of pseudo-random testing.

The number of the test patterns to be generated in the second phase might still be big to ensure required fault coverage. In BIST we need to store deterministic test patterns on a chip. So, the problem of violating the memory limitations might occur when storing deterministic patterns. Particularly this problem we want to solve in the thesis. We aim at obtainment of higher defect coverage of the first phase and significant decrease of the number of deterministic patterns in the second phase of mixed-mode BIST provided the use of P-PET in the scheme. Hence, less memory is needed to store the patterns.

The goal of the thesis is to encode the deterministic test patterns during P-PET. To accomplish the goal, we need to foresee the faults which would not be detected in the first phase of mixed-mode BIST and to test maximized number of these faults already in P-PET. Thus, we may not need mixed-mode BIST, and P-PET is enough to attain certain level of the fault coverage. This will give a significant improvement in terms of the memory needed to store the test patterns.

1.2 Thesis outline

The introductory part is followed by background information and state of the art in chapter 2. Then the formulation of the problem and the proposed solution are discussed in chapter 3. The described solution flow determines the order of the proceeding chapters. Firstly, the fault classification and deterministic test pattern generation will be considered in chapter 4. Then, we will deal with the process of selection of characteristic polynomials for P-PET in chapter 5. The experimental results will be discussed in chapter 6. Conclusions are made in chapter 7.

CHAPTER 2

PRELIMINARIES AND STATE OF THE ART

Contents

In this chapter we will discuss preliminaries necessary for understanding of the problem stated in the thesis and the solution proposed. Firstly, we will introduce basic definitions regarding testing of VLSI circuits as defect, fault, fault model and pattern. We will introduce fault model dependent testing and discuss its advantages and disadvantages. Then we will consider fault model independent testing with its pros and cons. And finally, the problem to be solved will be presented.

2.1 Basic Definitions

A **testing** of VLSI circuits is an experiment to check the presence of errors. It provides with information about the quality of the chip. Testing procedure consists of applying the set of inputs to the circuit and observing the outputs with subsequent evaluation of the outputs. The concept of testing is depicted in Figure 2.1.

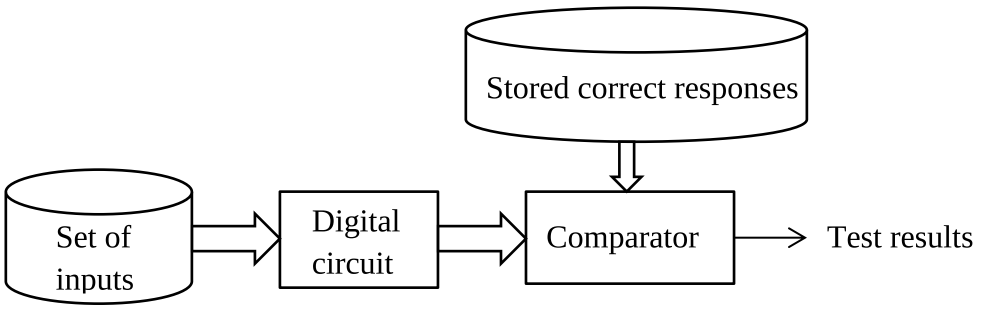


Figure 2.1: The concept of testing

A **defect** is a flaw or physical imperfection that may lead to a fault [6]. The purpose of the testing is to identify the presence of the defects in the circuit. We understand that the circuit has defects if we observe incorrect behavior.

The **sources of defect** are missing or extra material, wear-out, influence of environmental factors, such as temperature, humidity, vibrations causing the aging of components. Defects of interconnections among components can be subdivided to shorts and opens. Short happens when there is a connection which should not be, and open occurs as a result of breaking out of connection.

A **fault** is a representation of a defect reflecting a physical condition that causes a circuit to fail to perform in a required manner [6]. Faults can be classified to permanent, intermittent and transient depending on their duration. Permanent faults once occurred do not vanish. Intermittent faults appear for some periods of time. Transient faults arise one time and often are caused by environmental factors. Later on we will talk about permanent faults only.

A **fault model** is a logical representation of the fault. The fault model introduction makes analysis of the testing possible and also limits the scope of the faults to be considered. There exist different kinds of fault models. There are stuck-at fault model [34] (stuck-at “0” or stuck-at “1”) when there is a permanent short with ground or power; bridging fault model [12] in case of the short between two signal lines; CMOS transistor stuck-open [16] when transistor is stuck in the open state; and CMOS transistor stuck-short [20] when transistor is permanently shorted; memory faults [30] and delay faults [41].

Stuck-at fault model is the most popular fault model. It can be used for modeling many defects, it is technology independent, and it is simple. There are single stuck-at faults and multiple stuck-at faults. In the single stuck-at fault model, one line is assumed to be stuck-at “0” or “1”. For example, in the circuit depicted in Figure 2.2, the line x_3 is stuck-at “0”. In multiple stuck-at fault models, several lines are faulty at the same time. In order to identify the presence of the fault in a circuit, the test inputs which can detect the fault need to be applied. The set of inputs is composed of so-called patterns.

A **pattern** is an assignment of the inputs of the circuit under test. One can distinguish between **care bits** (or specified bits) which are either logic “0” or “1”, and **don’t care bits** (or unspecified bits) represented by “X”. In the case of care bits, specific input must be set to the value of the care bit in order to detect particular fault. In the case of don’t care bit, the input can be set either to “0” or “1” without influencing on the detection of the fault.

The pattern can detect the fault if the output of the faulty circuit is different from the output of the fault-free circuit when applying this pattern. In other words, if z is the output of the fault-free circuit and z_f is the output of the faulty circuit, then particular pattern tests the fault if $\boldsymbol{z \oplus z_f = 1}$.

Consider the combinational circuit consisting of primitive logic gates such as

AND, OR and NOT depicted in Figure 2.2. The circuit output function is $z = (x_1 + x_2) + \overline{x_2}x_3$. Assume that there is stuck-at "0" on the input line x_3. The faulty output function will be $z_f = x_2 + x_3$. Then, in order to detect this fault we need to have (2.1).

$$z \oplus z_f = (x_1 + x_2) + \overline{x_2}x_3 + (x_1 + x_2) = \overline{x_2}x_3 = 1 \tag{2.1}$$

So, to assure that $\overline{x_2}x_3 = 1$, it is necessary to have at the same time $x_2=0$ and $x_3=1$. The value of x_1 is not important, it can be either "0" or "1". Consequently, the pattern detecting the fault will be *X01*.

ATPG (Automatic test pattern generation/generator) is an electronic design automation technology to generate test patterns to be applied to the circuit in order to ascertain the presence or absence of the fault at some location of the circuit. In the process of ATPG, we can distinguish two main concepts. There are *fault activation* and *fault propagation*.

Consider the same circuit in Figure 2.2. Assume there is stuck-at "0" on the line y_2. We need to perform **fault simulation**, that is simulating of the circuit without this fault and with the presence of fault. Applying the test pattern 001, we get the output of the fault-free circuit $z=1$. The output of the faulty circuit is $z_f=0$. The test pattern 001 *activates* the fault stuck-at "0" on the line y_2 if it generates the error by creating the value on the line y_2 different from the value of assumed fault. And then the resulting signal value, or the fault effect, is propagated to the primary output. The propagation path in our example is y_2, y_3, z.

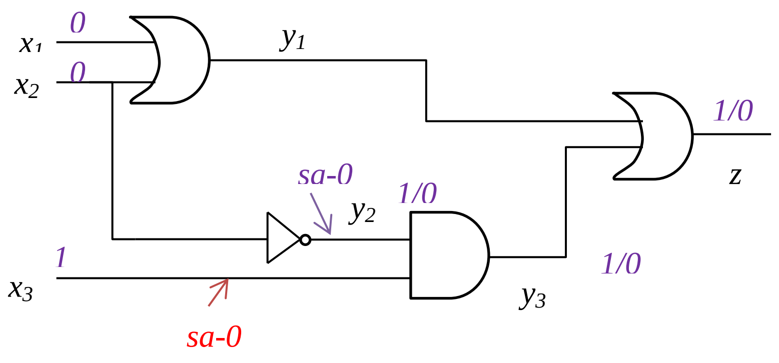


Figure 2.2: Combinational circuit

A **fault coverage** is a percentage of the faults detected after the test is performed. Fault coverage is determined by fault simulation. After performing the fault simulation, we have information regarding detected and undetected faults. Fault coverage is defined by equation (2.2). In order to calculate the fault coverage, we divide the number of detected faults to the total number of faults.

$$Fault\ coverage = \frac{\#detected\ faults}{Total\ \#\ faults} \tag{2.2}$$

The fault coverage evaluates the effectiveness and quality of the test. Fault coverage of 100% cannot be achieved if the circuit contains undetectable faults. For undetectable faults, it is not possible to find the pattern to differentiate the fault-free circuit from the faulty circuit. In this case, we will refer to effective fault coverage or test coverage calculated by formula (2.3). The effective fault coverage is calculated by division of the number of detected faults into subtracted by the number of undetectable faults total number of faults. The number of undetectable faults is also determined by fault simulation.

$$Effective\ fault\ coverage = \frac{\#\ detected\ faults}{Total\ \#faults - \#\ undetectable\ faults} \tag{2.3}$$

ATE (automatic test equipment) is equipment that performs testing using automation to perform measurements, obtain and evaluate the test results. ATE realizes the test by transporting and applying the test patterns and by evaluation of responses which are transported back to ATE. ATE is very precise equipment, but it is also very expensive. ATE is used for external testing. In internal testing the techniques called design for testability (DFT) are exploited.

DFT is a technique that ensures testability of the design. Testability of the design can be determined by controllability and observability.

Controllability is an ability to set some internal circuit nodes to specific values.

Observability is an ability to observe the values of internal nodes.

By increasing the level of testability of the design, the decrease in the test time and the cost can be reached in internal testing. On the other hand, additional hardware is required that affects the area overhead and causes delays. So, the balance is needed between introducing DFT and the gain which can be achieved by DFT.

Among DFT techniques should be mentioned ad hoc techniques [13] (initialization [13], test points insertion [6], partitioning of the circuit [13]), scan-based design [6] and boundary scan [34]. Scan design is widely used approach, method of distributing the test patterns to the internal circuit under test and responses to the outputs of the circuit by means of the scan chains.

A **scan chain** is a chain of registers or flip-flops allowing easy to read and write the content. The test patterns are shifted in via the scan chain and results are shifted out to the chip output pins.

One way of application of DFT is built-in self-test (BIST). In BIST, the part of the circuit tests itself. BIST scheme makes testing easier, faster and cheaper. In BIST, additional hardware and software are needed to enable self-checking.

Having discussed the basic terms, we can introduce typical BIST scheme. The typical BIST application hardware includes test pattern generator (TPG), response analyzer (RA), circuit under test (CUT) and controller (Figure 2.3).

It should be noted that in BIST not only the test pattern generator is on the chip, but also the response analyzer. The whole process of testing is under regulation of controller. In this scheme, TPG generates the test patterns to be applied to CUT; the outputs of CUT are evaluated by RA by comparing them to the correct outputs. It is of great importance the way TPG generates the patterns since it affects the fault coverage, test time and test cost.

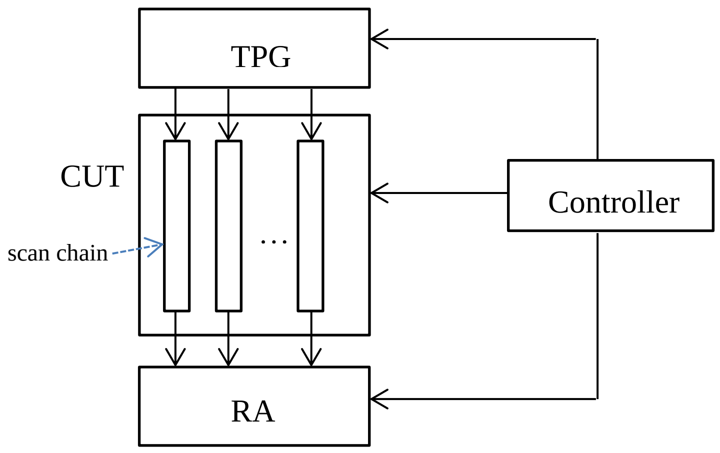


Figure 2.3: General BIST hardware scheme

Testing can be based on fault model or can be independent of fault model. In relation to that, testing is classified to *fault model dependent* testing and *fault model independent* testing.

2.2 Fault model dependent testing

Fault model dependent testing is entirely based on the assumed fault model. It often results in a shorter test length. Fault model dependent testing has limitations of fault model since fault model cannot precisely reflect the nature of the fault. In addition to that, in order to perform thorough test of the circuit, one need to consider several fault models that increases the test length and time. Another disadvantage of fault model dependent schemes is that they often require fault simulation to be performed which is expensive. Test pattern generation for fault model dependent testing can be performed by ATPG. In this case, ATPG generates patterns for the considered list of the faults of particular fault model.

We will consider fault model dependent testing on the example of *pseudo-random testing*. In pseudo-random testing, patterns have many characteristics of random patterns, but they are generated deterministically, that is why the method is called pseudo-random. If we have the circuit with *n* inputs, pseudo-random TPG generates the subset of 2^n input combinations.

Pseudo-random patterns can be generated by cellular automata or linear feedback shift register (LFSR). Cellular automata have better randomness property. Cellular automata consist of the cells with forward and backward connections. If we denote by C_i the state of the current cell, then the next state of C_i will be determined by (C_{i-1}, C_i, C_{i+1}), where C_{i-1} and C_{i+1} are neighboring cells. More information regarding cellular automata can be found in [35].

We will concentrate on using LFSR as a pattern generator for pseudo-random testing. For that, we need basic knowledge of the LFSR. Mathematical background of LFSR is presented in [36]. There are two types of the LFSR depending on where XOR-gates are placed. In Figure 2.4 the external type LFSR is presented, and in Figure 2.5 the internal type LFSR is depicted.

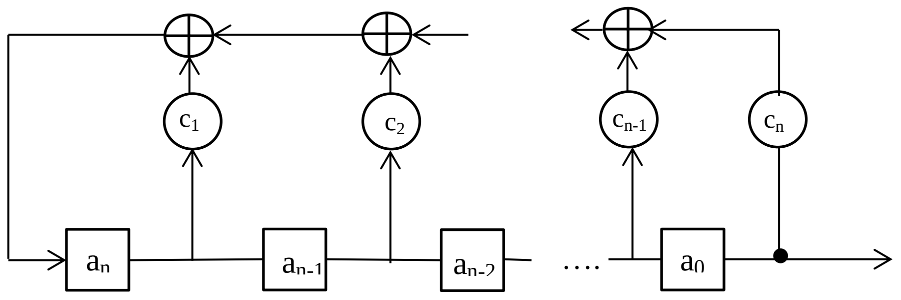


Figure 2.4: External type linear feedback shift register

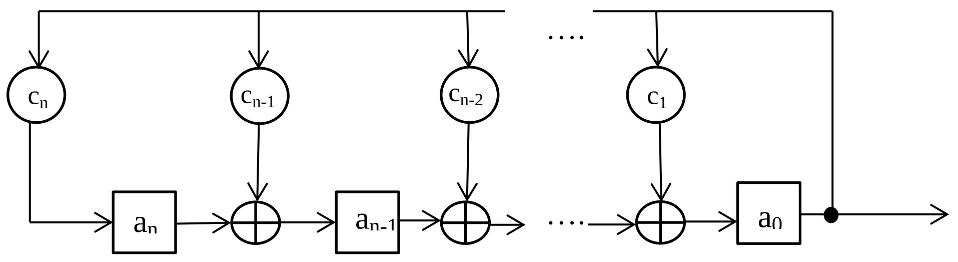


Figure 2.5: Internal type linear feedback shift register

A LFSR consists of the number of D-type flip-flops $a_0 \ldots a_n$ and exclusive OR (XOR) gates. The configuration of the LFSR is determined by the feedback or characteristic polynomial $P(x) = 1+c_1x+c_2x^2+\ldots+c_nx^n$ where $c_1, c_2, \ldots, c_n$ are feedback coefficients of the value '0' or '1'signifying the presence or absence of connection. The degree of characteristic polynomial is equal to the number of flip-flops of the LFSR. The LFSR generates periodic sequences with maximal period of 2^n-1 where *n* is the degree of characteristic polynomial.

A **primitive polynomial** is a characteristic polynomial of the LFSR generating maximum-length sequence. Primitive polynomial *p(x)* has the property that if we compute the remainders of the polynomial division of increasing powers of *x* to *p(x)* (i.e., *x modulo p(x)*, x^2 *modulo p(x)*, x^3 *modulo p(x)* and so on), we will obtain all possible non-zero polynomials of degree less than *p(x)*.

A **seed** is an initial state of a LFSR. On the next cycle all seed elements are shifted by one position and new seed element is calculated by applying XOR operation on the seed elements from the previous cycle specified by the feedback polynomial.

Consider the LFSR depicted in Figure 2.6. The feedback polynomial of this LFSR is $P(x) = x^4 + x^1 + 1$. *P(x)* is a primitive polynomial. The degree of this primitive polynomial is equal to 4. We will run this LFSR for $(2^4-1) = 15$ cycles and fill in Table 2.1.

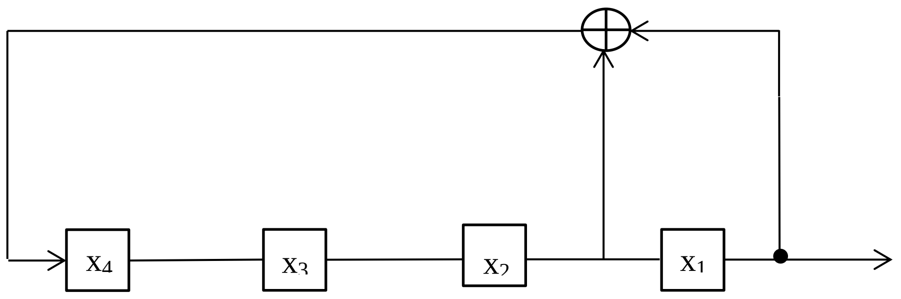


Figure 2.6: Example of external linear feedback shift register

Clock cycle				
0	x_4	x_3	x_2	x_1
1	$x_2+ x_1$	x_4	x_3	x_2
2	$x_3+ x_2$	$x_2+ x_1$	x_4	x_3
3	$x_4+ x_3$	$x_3+ x_2$	$x_2+ x_1$	x_4
4	$x_2+ x_1+ x_4$	$x_4+ x_3$	$x_3+ x_2$	$x_2+ x_1$
5	$x_3+ x_1$	$x_2+ x_1+ x_4$	$x_4+ x_3$	$x_3+ x_2$
6	$x_4+ x_2$	$x_3+ x_1$	$x_2+ x_1+ x_4$	$x_4+ x_3$
7	$x_2+ x_1+ x_3$	$x_4+ x_2$	$x_3+ x_1$	$x_2+ x_1+ x_4$
8	$x_4+x_3+ x_2$	$x_2+ x_1+ x_3$	$x_4+ x_2$	$x_3+ x_1$
9	$x_4+ x_3+ x_2+ x_1$	$x_4+x_3+ x_2$	$x_2+ x_1+ x_3$	$x_4+ x_2$
10	$x_4+ x_3+ x_1$	$x_4+ x_3+ x_2+ x_1$	$x_4+x_3+ x_2$	$x_2+ x_1+ x_3$
11	$x_4+ x_1$	$x_4+ x_3+ x_1$	$x_4+ x_3+ x_2+ x_1$	$x_4+x_3+ x_2$
12	x_1	$x_4+ x_1$	$x_4+ x_3+ x_1$	$x_4+ x_3+ x_2+ x_1$
13	x_2	x_1	$x_4+ x_1$	$x_4+ x_3+ x_1$
14	x_3	x_2	x_1	$x_4+ x_1$
15	x_4	x_3	x_2	x_1

Table 2.1: Sequences generated by LFSR

The values from the last column fill in the scan chain. In Table 2.1 the shifting can be easily observed and calculation of new element by applying XOR operation on the forth and the third column. The values on the 15th cycle are the same as initial. This proves that *P(x)* is indeed the primitive polynomial.

LFSR tends to generate equal number of “0”s and “1”s. For some circuits the biased distribution of “0”s and “1”s is preferred. This is explained by the presence of **random pattern resistant faults**. Random pattern resistant faults have low detection probability; they are detected by only a few patterns. As a result, the presence of random pattern resistant faults negatively influences the fault coverage.

There exist several ways to tackle this problem. Firstly, in order to reduce random pattern resistivity, weighted TPG can be applied. And, secondly, the fault coverage can be improved by mixed-mode BIST scheme.

In **weighted TPG** ([34], [42]) the distribution of “0”s and “1” in the pattern depends on the weight. Different weights should be pre-computed for the circuitry since for each part of the circuit specific distribution has better results. As TPG, LFSR with combinational logic is used [29].

2.2.1 Mixed-mode BIST

Mixed-mode BIST scheme consists of two phases. The first phase is pseudo-random testing during which the large part of the faults is tested, and the second is applying deterministic patterns for the random pattern resistant faults which were not detected in the first stage.

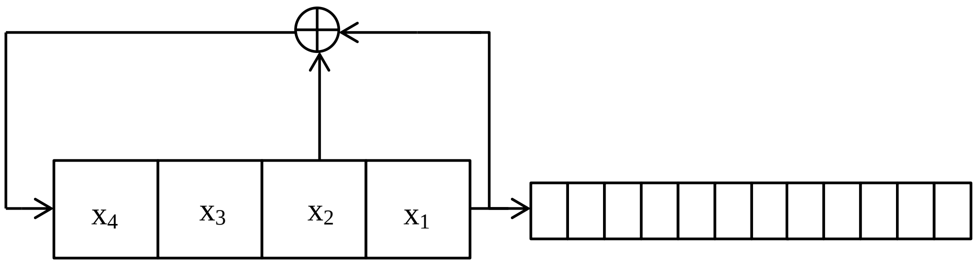
For the purpose of deterministic pattern generation, automatic test pattern generator is used. All deterministic patterns need to be stored in a ROM (Read-only memory). The size of the pattern set directly influence the size of the memory. Sometimes the size of the pattern set to be stored violates the memory requirements. In order to reduce the memory needed to store the patterns, compression techniques are used.

As we already know, the pattern consist of care bits, which are important for particular fault detection, and don’t care bits, which can be either “0” or “1”. Compression technique takes advantage of don’t care bits assigning to them certain logic values which enables compression. Or, in other words, compression mechanism looks for the patterns which differ in a small number of bits and tries to combine them.

Different approaches have been proposed for the test data compression. They can be classified to code-based schemes (e.g., Dictionary code [5], Huffman code [3]), linear-decomposition-based schemes (e.g., linear LFSR reseeding based schemes), broadcast-scan-based schemes [6]. In mixed-mode BIST, the deterministic test patterns can be encoded in the sequence of pseudo-random patterns by bit-flipping or bit-fixing approaches [37], [38], [1]. We will consider in more details LFSR reseeding as very popular and elegant solution for compression.

In LFSR reseeding [1], deterministic test patterns are encoded into seeds. After initializing the LFSR with pre-computed seed, LFSR generates deterministic test pattern and fills in the scan chain with the pattern after *m* cycles, where *m* is the length of the scan chain. A seed can be computed by solving the system of linear equations based on the feedback polynomial of LFSR, where each specified bit of the pattern is represented by an equation in terms of the seed variables [7].

In Figure 2.7 the LFSR with feedback polynomial $P(x) = x^4 + x^1 + 1$ is depicted. We start with running the LFSR with seed elements as variables and filling in the scan chain.



TP={X, 0, X, X, X, 0, X, X, X, X, 1, 1}

Figure 2.7: LFSR with scan chain

Then we compose the system of linear equations according to the bits of the given pattern. The system of linear equations for the test pattern *TP* is represented by (2.4).

$$\begin{cases} x_1 = 1 \\ x_2 = 1 \\ x_3 + x_4 = 0 \\ x_1 + x_2 + x_3 = 0 \end{cases} \tag{2.4}$$

In our example, the system of equations (2.4) has the solution: x_1=1, x_2=1, x_3=0, x_4=0 which is the seed needed to encode the pattern *TP*. The number of care bits in the pattern determines the number of equation. The less number of care bits in the pattern, the more probability to find the seed.

In [1] Könemann showed that single-polynomial LFSR should have the length of *(s+20)* in order to find the seed for the pattern with *s* care bits with the probability less than 10^{-6}. Then multiple-polynomial LFSRs [2] (MP-LFSR) were introduced. Using MP-LFSR, test pattern with *s* care bits can be encoded with *(s+4)* bits.

MP-LFSR is an LFSR with reconfigurable feedback network, where AND-gates are used to control the feedback polynomial.

Multiple-polynomial LFSR gives more freedom and flexibility during pattern encoding in the sense that hardware is programmable with the seed, and polynomial information can be modified [7].

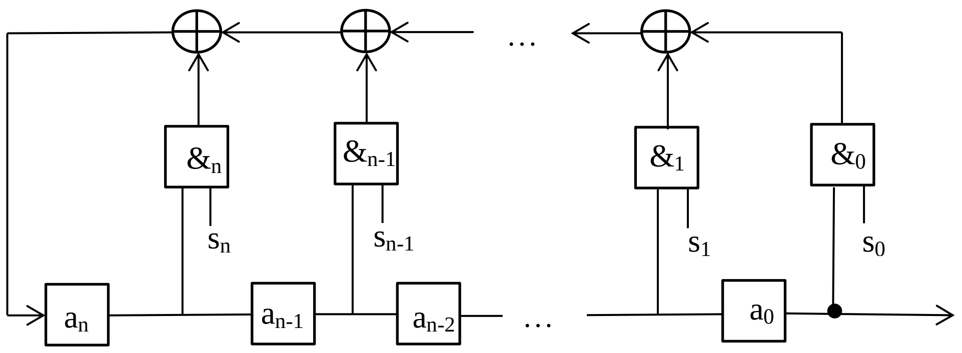


Figure 2.8: Multiple-polynomial LFSR

In Figure 2.8 the MP-LFSR is illustrated. There the sequence of bits s_0, s_1,…, s_n are used to determine the resulting feedback polynomial. If the bit is “0”, the output from AND-gate is also “0”, and corresponding branch is not present. As we can see in Figure 2.8, the hardware overhead in comparison with the single-polynomial LFSR consists of only AND-gates.

In Figure 2.9 the encoding scheme based on the reseeding of the LFSR is presented. The seeds and corresponding polynomial identifiers are stored in a ROM. Polynomial identifier is used to configure the feedback network of the LFSR.

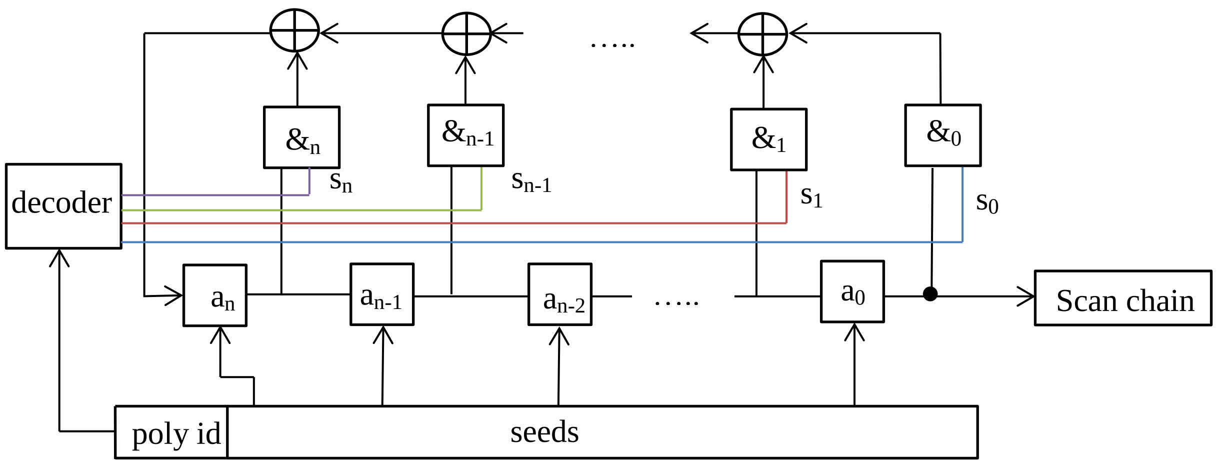


Figure 2.9: LFSR reseeding scheme

The basic hardware scheme for BIST is Self-Test Using Multiple Input Signature Register (MISR) and Parallel Shift Register Sequence generator (PRSG) (STUMPS), and in the work presented we will also use this scheme.

The general STUMPS scheme is illustrated in Figure 2.10. In STUMPS, PRSG

[46] is used for the test pattern generation. Multiple scan chains are applied for the purpose of minimization of the test time. Scan chains may have different lengths. In this scheme, MISR is used to compact responses [33] since the outcome of CUT should be compacted, otherwise processing of huge amount of data becomes impossible. In this case, the circuit is tested by comparing the output signature with the correct pre-computed signature. For this purpose the signature, that is the compressed form of the response, is calculated. Further information regarding response compaction can be found in [25], [26], [27].

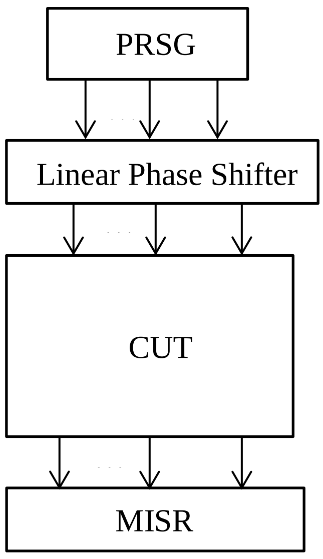


Figure 2.10: General STUMPS scheme

PRPG loads the scan chains in parallel. The system clocks are triggered. Then the test responses are shifted to the MISR, at the same time new patterns are shifted in. To improve the randomness of the PRPG, linear phase shifter may be used [46].

2.3 Fault model independent testing

Fault model independent testing does not have the limitations of the fault models. This kind of testing may result in a long test length because it does not target specific faults. One example of fault model independent testing is exhaustive testing.

In exhaustive testing, all possible 2^n input combinations are applied to the circuit with *n* inputs. Consider the circuit under test in Figure 2.11. This circuit has six inputs and three outputs. One should apply $2^6 = 64$ test patterns to test the circuit exhaustively.

Test patterns for exhaustive testing can be generated by binary counters or LFSRs. Since the order of the patterns generated is not important, it is better to use LFSRs [39].

LFSRs require less hardware than binary counters.

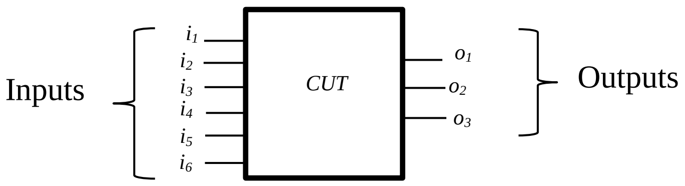


Figure 2.11: Circuit under test

If the number of inputs is small, then exhaustive testing is very useful because 100% of all combinational defects are guaranteed to be detected. But in the case of the large number of inputs, exhaustive test ends up with long test length which is infeasible. In order to have the benefits of exhaustive testing and less number of the test patterns, pseudo-exhaustive testing was proposed.

Pseudo-exhaustive testing (PET) [31], [43] exercises different kinds of circuit segmentation. Segment is a subcircuit. Each segment is tested exhaustively. In pseudo-exhaustive testing, one can achieve complete detection of combinational defects. The advantage of pseudo-exhaustive testing is that it requires less number of the test patterns than exhaustive testing. Segmentation of the circuit can be logical (cone and sensitized path segmentation) and physical (hardware) [13].

We will start with consideration of the cone segmentation. This approach is also called *verification testing* [15].

An **output cone** is a sub-circuit containing all structural predecessors of one primary output.

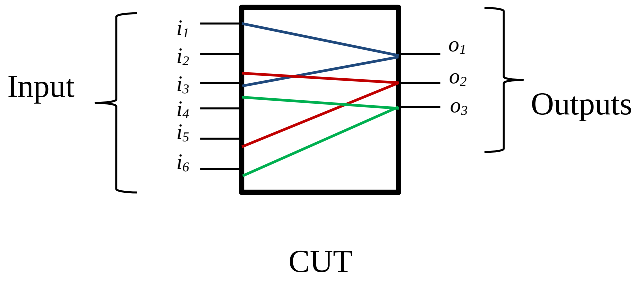


Figure 2.12: Cone segmentation

In the case of the cone segmentation, each cone is tested exhaustively, and all cones are tested concurrently. If we have *m*-outputs circuit, than this technique defines *m* cones. The cone segmentation example is shown in Figure 2.12.

The **size of the cone** is determined by the number of the inputs in the cone. Consider the circuit in Figure 2.12; it has six inputs and three outputs. We divided the circuit into three cones (by the number of outputs). The size of each cone is equal to three. To test the circuit we will need $3*2^3=24$ patterns instead of *64* patterns as in exhaustive testing.

Verification testing cannot be feasible if the circuit has outputs depending on too many circuit inputs [43]. If the circuit is too big and complex, three other techniques are used: sensitized path segmentation, hardware partitioning and partial hardware partitioning.

To start off with, we will consider sensitized path segmentation [31]. A line in a circuit whose value in the test changes in the presence of fault is said to be **sensitized** to the fault by the test**.**

Sensitized path is a path composed of sensitized lines.

In sensitized path segmentation, some inputs are set to the values, so that particular segments are triggered. Consider combinational circuit in Figure 2.13. C_1 and C_2 are sub-circuits. In order to test C_1 exhaustively, one need to apply 2^{n_1} test patterns to *a*. At the same time we need to set *b* to some value so that $d=1$. Sensitized path in this case is *c-f*. C_2 can be tested in a similar way. Totally, we will need $2^{n_1} + 2^{n_2} + 1$ patterns to test the circuit [13] instead of $2^{n_1+n_2}$ patterns necessary for exhaustive testing.

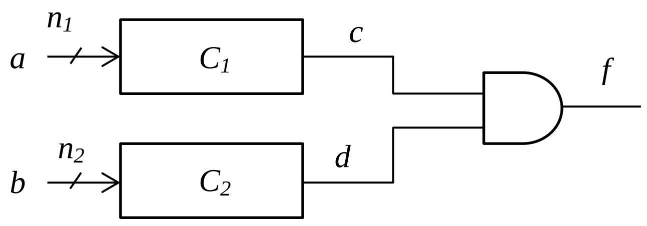


Figure 2.13: Testing via sensitized path segmentation [13]

Sensitized path segmentation technique requires high computational time. It may result in incomplete defect coverage because of the possible defects presence between the segments [14].

In hardware partitioning technique [31], [28], hardware is added to the circuit to control and observe the segment’s inputs and outputs. So, the circuit is divided into small sub-circuits. Each subcircuit is directly controllable and observable. One way is to use multiplexer partitioning. The disadvantage of adding extra hardware is in the speed decrease and the cost of implementation.

In partial hardware partitioning [44], circuit's inputs indirectly control the segment's input, and added hardware directly observes the segment output. Partial hardware partitioning has features of hardware partitioning and sensitized partitioning. Test patterns are applied to the primary inputs as in the sensitized partitioning. But instead of sensitizing the segment outputs to circuit outputs, extra hardware is added to allow the segments outputs to be observed directly as in the hardware partitioning. This approach results in lower hardware overhead and less test set generation complexity [44].

There are number of ways to generate test patterns in PET: syndrome-driver counter [19], constant-weight counter [15], combined LFSR and shift register [21], combined LFSR and XOR gates, condensed LFSRs [22], cyclic LFSRs [23], [24]. These methods in some cases may result in a long test length, which is not applicable for today's circuits.

Summing up, PET results in a less number of test patterns than exhaustive testing, and it is fault model independent. PET attains mentioned benefits by means of logical or hardware segmentation. If the circuit is very large, cone segmentation has infeasible test length. In order to tackle this problem, hardware segmentation was proposed. The disadvantages of hardware segmentation technique are in the additional delays and in the cost increase. In order to have the benefits of cone segmentation (no hardware overhead) and make it feasible for larger circuits, partial pseudo-exhaustive testing was proposed.

2.4 Partial Pseudo-Exhaustive Testing

Partial pseudo-exhaustive testing (P-PET) [9] is a technique that is based on the cone segmentation. In P-PET, only cones up to specified *MAX* size are tested exhaustively. So, the problem of long test length is tackled there. Cone segmentation technique becomes feasible.

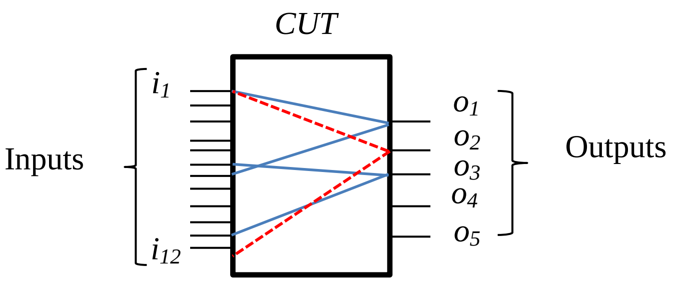


Figure 2.14: Cone segmentation for P-PET

Consider the circuit in Figure 2.14. If we assume that *MAX=6*, then we start with exhaustive testing of the blue cones of the size 6. The red cone will not be tested exhaustively since the size of it is larger than 6.

After testing exhaustively the cones up to *MAX* size, some faults from the larger cones will not be detected. It should be noted that for the faults from the larger cones, P-PET behaves like pseudo-random testing. In order to target the faults from the larger cones which were not detected by P-PET, deterministic patterns need to be generated. So, P-PET cannot be used alone but in a mixed-mode BIST scheme where in the first phase is P-PET instead of pseudo-random testing. Consequently, P-PET scheme is based by half on fault model independent testing (first phase of mixed-mode BIST) and by other half on fault model dependent testing (second phase of mixed-mode BIST).

Consider STUMPS architecture depicted in Figure 2.15 for P-PET testing. Pattern generation for P-PET is implemented by MP-LFSR. The corresponding characteristic polynomials are stored in a ROM to update MP-LFSR. So, in the first stage of the mixed-mode BIST, MP-LFSR, configured by stored characteristic polynomials, generates test patterns to tests all cones up to *MAX* size exhaustively.

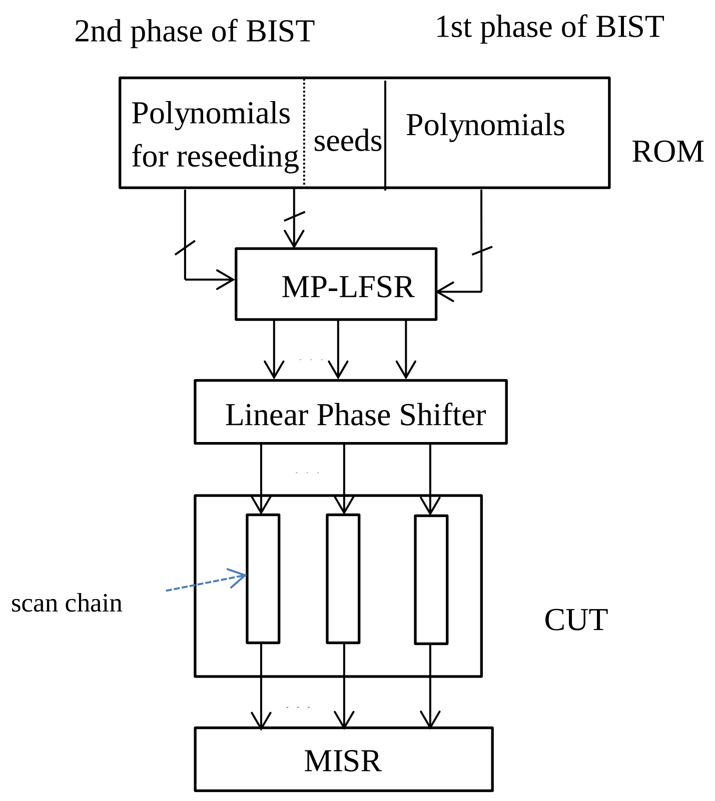


Figure 2.15: STUMPS architecture for P-PET testing

Then for the faults from the larger than *MAX* cones which were not detected by the first stage, deterministic test patterns are encoded by LFSR reseeding technique. Seeds and characteristic polynomials are also stored in a ROM. In the second phase of the mixed-mode BIST, the deterministic test patterns generated by MP-LFSR configured by corresponding characteristic polynomial and initialized with particular seed are applied.

Characteristic polynomials for MP-LFSR are pre-calculated from the circuit structure. The process of choosing feedback polynomials for P-PET is NP-complete problem, and set covering heuristics is applied to solve the problem efficiently. In [9] the goal was set to obtain the smallest possible number of feedback polynomials. In this case, the test application time will be minimal, and less memory is need to store the polynomials.

Formulating the problem from [9] more formally, let K_c be the set of cones of the circuit up to the size *MAX*. Let *P* be the set of polynomials up to degree *MAX*. The aim is to find the smallest set of polynomials P_s, the subset of the set *P*, which will cover the set of the cones K_c (2.2).

$$\forall\, k \in K_c\ \exists\, p \in P_s\ \ p \prec k \tag{2.2}$$

We will use the following designation $p \prec k$ to denote that the polynomial *p* tests, or covers, the cone *k*. It is said that the feedback polynomial of degree *r* tests the cone *k* which has *s* inputs when 2^r-*1* different patterns generated cover 2^s-*1* different assignments of the cone. On the other hand, this approach is very time consuming and not applicable for the large set of polynomials.

For this task *Barzilai's Theorem* [32] is very helpful.

Barzilai's theorem:

Let $(a_\tau)_{\tau\geq 0}$ be a shift register sequence generated by a primitive polynomial *p* of degree *r*. The set T=$(a_0,...,a_{r-1})$, $(a_1,\dots,a_r)$, …,$(a_{2^r-2}, a_0,\dots,a_{r-2})$ is an exhaustive enumeration of the assignment of $(i_1,\dots, i_s)$, if the remainder classes (X^{i1} mod *p*), …,(X^{is} mod *p*) over GF(2) are linearly independent. Consequently, to determine whether the polynomial covers the cone we need to check the linear independency of remainders (X^{i1} mod *p*), …,(X^{is} mod *p*) [32].

Figure 2.16 shows an example for checking whether particular feedback polynomial can cover the cone.

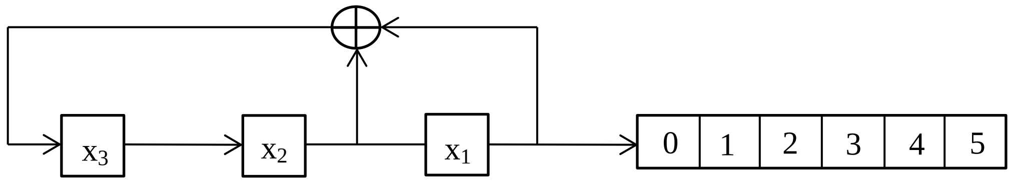


Figure 2.16: LFSR and scan chain

Depicted in Figure 2.16 LFSR, has feedback polynomial $P(x) = x^3+x^2+1$. In this example, we are considering the cone *k = {0, 2, 4}*. So, according to Barzilai's Theorem, we need to calculate remainders (2.5).

$$(x^0) \; mod \; (x^3+x^2+1) = (1)$$
$$(x^2) \; mod \; (x^3+x^2+1) = (x^2) \qquad (2.5)$$
$$(x^4) \; mod \; (x^3+x^2+1) = (x^2+x+1)$$

The remainders *(1), (x^2), (x^2+x+1)* are linearly independent, consequently, the feedback polynomial $P(x) = x^3+x^2+1$ covers the cone *k*.

If we consider the cone k_1*= {0, 2, 4, 5}*, the corresponding remainders presented in (2.6) are linearly dependent.

$$(x^0) \; mod \; (x^3+x^2+1) = (1)$$
$$(x^2) \; mod \; (x^3+x^2+1) = (x^2) \qquad (2.6)$$
$$(x^4) \; mod \; (x^3+x^2+1) = (x^2+x+1)$$
$$(x^5) \; mod \; (x^3+x^2+1) = (x+1)$$

Consequently, the polynomial $P(x) = x^3+x^2+1$ does not cover the cone k_1.

Having discussed the principles of P-PET, we continue with advantages of P-PET. Usually in the first phase of the mixed-mode BIST, pseudo-random test is used. Applying P-PET in the first phase of the mixed-mode BIST results in the increased defect coverage and lower number of deterministic patterns in the second phase of mixed-mode BIST.

The problem with this realization may arise when in the second phase of mixed-mode BIST, one need to encode large number of deterministic patterns and store them on a chip. Memory requirements may limit the storage for deterministic patterns. That is the reason why we want to tackle this problem and try to target some part of the faults from the larger than *MAX* cones which cannot be covered by P-PET already during P-PET stage. Or, in other words, we want to encode the deterministic patterns, which would be generated in the second phase of the mixed-mode BIST for the uncovered faults from the larger cones, in P-PET. So, the defect coverage of P-PET will be increased. This can be done by appropriate selection of the feedback polynomials for P-PET pattern generation. In some cases, only P-PET would be enough to reach certain fault coverage, in others- less number of patterns need to be encoded and stored.

CHAPTER 3

PROBLEM FORMULATION AND SOLUTION FLOW

Contents

In this chapter we will provide detailed problem description and formulation of the goal to be reached in the thesis. We will also discuss the proposed solution flow to achieve the objective.

3.1 Problem formulation

In mixed-mode BIST scheme, two phases are distinguished. In the first phase, patterns generated on a chip are applied to the circuit. For the faults which were not detected in the first phase, deterministic test patterns are stored on a chip. In the second phase, these stored patterns are used to reach the certain fault coverage.

P-PET in the first phase of mixed-mode BIST tests exhaustively cones up to some *MAX* size. In Figure 3.1 the cones of the circuit up to *MAX* size are shown with the blue color. For the faults from the larger than *MAX* cones, P-PET behaves like pseudo-random test. In Figure 3.1 the possible locations of these faults are represented by yellow areas.

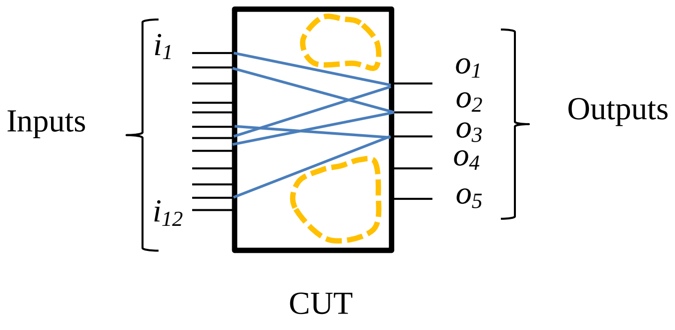


Figure 3.1: Cone segmentation in P-PET

The majority of the faults are already covered by P-PET. For the faults not detected by P-PET, deterministic test patterns need to be generated. One possibility is to store deterministic test patterns on a chip. Another way is to encode these patterns in P-PET. The degree of freedom to choose primitive polynomials to cover all cones up to *MAX* size can be exploited. So, our goal can be formulated as follows: we aim to select polynomials for P-PET pattern generation such that the set of undetected faults is minimized (or the number of detected faults from the larger than *MAX* cones is maximized).

The goal formulation can be further concretized. We can either aim to have the same number of polynomials as in P-PET where only cones up to *MAX* size are targeted, or to obtain higher fault coverage by means of adding extra polynomials. So, we can distinguish two subgoals:

1) Selection of minimal number of polynomials for P-PET so that the number of undetected faults is minimized.
2) Selection of polynomials for P-PET so that the number of undetected faults is minimized (the number of polynomials can be more than minimal)

We will define the problem more formally.

Given the circuit to be tested, we denote K_{MAX} the set of cones of the circuit up to the size *MAX*. The set of cones larger than *MAX* is denoted as K_{LMAX}, and F stands for the set of the faults from K_{LMAX}. We also have the set of polynomials up to degree *MAX*, P_{MAX}. Let $K_p = \{k \in K_{MAX},\ p \prec k\}$ be the set of cones from K_{MAX} tested by the polynomial p.

The aim is to find the set of polynomials $P_s \subset P_{MAX}$ such that $K_{MAX} = \bigcup_{p \in P_s} K_p$, and the set of undetected faults F_{ud} is minimized.

3.2 Solution flow

We aim to target the faults from the set F_{ud} during the selection of polynomials for P-PET. But the genuine set of the faults F_{ud} can be obtained only after applying P-PET. We can exploit the fact that P-PET for the set of faults F from the larger than *MAX* cones behaves like pseudo-random test and classify the faults from the set F into hard-to-detect faults (not detectable by applying pseudo-random patterns) and easy-to-detect faults (detectable by pseudo-random pattern sequence). This fault classification can be performed by fault simulation of the large number of pseudo-random patterns (for example, 10^6 patterns). It is important to note that the set of hard-to-detect faults will be a superset of the set F_{ud} since the number of patterns applied in the multiple-polynomial scheme of P-PET is larger than 10^6.

Having obtained the list of hard-to-detect fault, it is necessary to encode these faults by test patterns so that we can take them into account while selecting the feedback polynomials for P-PET. In the step of checking whether some polynomial can cover the pattern, we will use the idea of MP-LFSR reseeding presented in the second chapter. The polynomial covers the pattern, if there is such a seed that LFSR will generate this particular pattern, given the configuration of the LFSR represented by certain feedback polynomial. It is necessary to emphasize that the less number of care bits the test pattern has, the bigger the probability to find the polynomial covering the pattern. This was discussed in subsection 2.2 of chapter 2. So, for us it is important to generate test patterns with minimized number of care bits.

Given the set of cones up to *MAX* size, the set of polynomials up to degree *MAX*, the pattern list with minimized number of care bits, we aim to select the polynomials, so that they cover all the cones up to the size *MAX*, and the number of deterministic test patterns covered is maximized. This is a set covering problem where heuristics is applied to solve it efficiently.

To sum up, the following steps will be taken to accomplish the goal:

1) Fault classification on hard-to-detect and easy-to-detect faults
2) Generation of the test patterns for hard-to-detect faults with minimized number of the care bits
3) Set covering heuristics to choose feedback polynomials for P-PET TPG

In chapter 4 and 5, these steps are described in details.

CHAPTER 4

FAULTS CLASSIFICATION AND TEST PATTERN GENERATION

Contents

In this chapter we will consider the first two steps necessary to achieve the formulated goal. In the first subsection, we will deal with fault classification and obtainment of the list of hard-to-detect faults. In the second subsection, we will discuss test pattern generation process with minimized number of care bits for the list of hard-to-detect faults.

4.1 Fault classification on hard-to-detect and easy-to-detect faults

In this subsection, we will discuss in details the first step in the process of achieving the goal described in chapter 3. In general P-PET, all faults from the cones up to *MAX* size are detected. For the faults from the larger than *MAX* cones, P-PET acts like pseudo-random test. Hence, some faults from the larger than *MAX* cones are also detected, some of them are not detected. We need to obtain a safe approximation to the set of the faults F_{ud} not detected by P-PET.

We will approach the problem by classification of the faults from set *F*, the set of the faults from the larger than *MAX* cones, into detectable by pseudo-random test sequence (easy-to-detect faults) and not detectable by pseudo-random test (hard-to-detect faults).

This classification is achieved by performing the fault simulation when applying the large number of random patterns. Figure 4.1 illustrates the idea. Given the circuit netlist, the list of faults from the set *F* of stuck-at model, we simulate the circuit in the presence of a million of random patterns. The resulting list of undetected faults will be equivalent to the list of hard-to-detect faults. All detected by the fault simulation faults are identical to the easy-to-detect faults.

The obtained set of hard-to-detect faults is a super set of the set of faults F_{ud}. This is due to the fact that the number of patterns applied in multiple-polynomial scheme of P-PET is larger than a million, the number of random patterns we applied for the fault simulation.

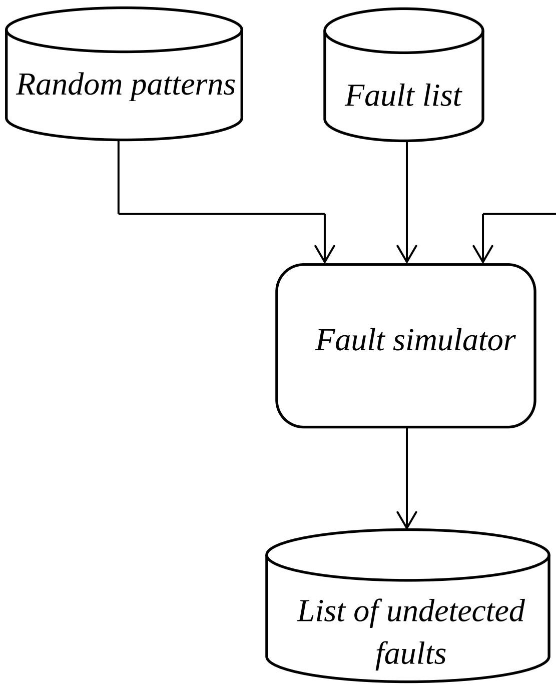


Figure 4.1: Fault simulation

The number of random patterns we applied in the fault simulation was a million. We will discuss now the relation between the number of random patterns applied and the fault coverage in pseudo-random testing.

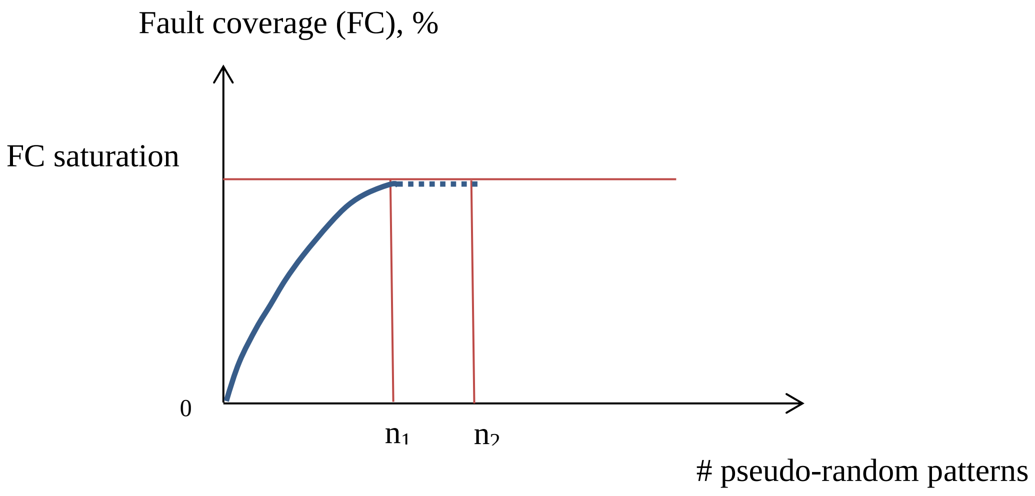


Figure 4.2: Fault coverage of pseudo-random testing

In Figure 4.2 the dependency of the fault coverage from the number of pseudo-random patterns applied in pseudo-random testing is shown [40]. The fault coverage rises in logarithmic manner towards the saturation.

According to Figure 4.2, in the beginning the slope of the curve is quite steep till it reaches its saturation. So, the fault coverage is growing rapidly until it saturates [40]. Hence, it does not matter how many patterns to apply n_1 or n_2 when we are close to the saturation of the fault coverage.

The fault simulation was performed for industrial circuits of different sizes. The results of the fault simulation and discussions are presented in chapter 6. The percentage of hard-to-detect faults from the faults of the set *F* lies in the range from 0.6% to 50%.

4.2 Generation of deterministic test patterns for hard-to-detect faults

Having obtained the list of hard-to-detect faults, we aim now at encoding these faults into test patterns with minimized number of care bits. The requirement of minimized number of care bits is very essential. The patterns generated for hard-to-detect faults will be used further in the process of selection of characteristic polynomials. And the less number of care bits the pattern has, the greater the probability for the polynomial to cover the pattern.

We will consider several methods for the test pattern generation: ATPG with minimum number of care bits and commercial ATPG.

4.2.1 ATPG with minimum number of care bits

The process of the test pattern generation with minimum number of care bits will be based on satisfiability (SAT) problem. To start off with, we will introduce basic definitions about SAT taken from [45].

A **Boolean formula** is a logic expression defined over variables that take value “True” or “False” which are identified with “0” or “1”.

A **truth assignment** to a set *V* of Boolean variables is a map $\sigma: V \to \{0,1\}$.

A **satisfying assignment** for *F* is a truth assignment σ such that *F* evaluates to 1 under σ.

Conjunctive normal form (CNF) of propositional formulas F is a conjunction (AND, $\wedge$) of clauses, where each clause is a disjunction (OR, $\vee$) of literals, and each literal is either a variable or its negation (NOT, ¬).

$$F = (a \vee \neg b) \wedge (\neg a \vee c \vee d) \qquad (4.1)$$

Expression (4.1) shows the example of CNF. There CNF formula with four variables and two clauses is presented.

The Boolean SAT problem can be formulated as follows: given a CNF formula *F*, does *F* have a satisfying assignment? And often the problem includes finding of an actual satisfiability assignment.

We will consider first SAT-based two-valued ATPG. SAT formulas are based on two values: “0” and “1”. In Figure 4.3 AND-gate with two inputs (*a* and *b*) and one output (*s*) is depicted. The SAT clause in CNF for this AND-gate is represented by (4.2). Indeed, for all possible assignments of *a, b, s* from the truth table (Table 4.1) *f* evaluates to ‘True’.

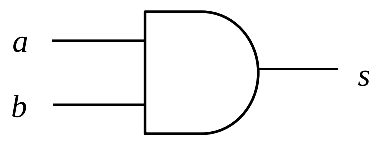


Figure 4.3: AND-gate for 2-valued logic

$$f = (a+\neg s)(b+\neg s)(\neg a+\neg b + s) \qquad (4.2)$$

a	b	s
0	0	0
0	1	0
1	1	1
1	0	0

Table 4.1: Truth table for AND-gate

Using SAT-based two-valued ATPG we will be able to generate patterns only with two values “0” and “1”. Whenever there is a freedom of assignment the value to the literal, the default value of “0” or “1” will be assigned. Since we aim at the least possible number of care bits in the pattern, we need to have patterns with a lot of don’t care bits “X”. Therefore, three-valued logic, “0”, “1” and “X”, is needed instead of two-valued logic.

For each of the primitive gates, the new SAT clauses were generated based on three-valued logic. To extend the logic, it was assumed that ‘X’ is represented by ‘00’, '0'- by '01', '1'- by '11', and the last combination is forbidden. To get the clauses, the truth table for each primitive gate was composed.

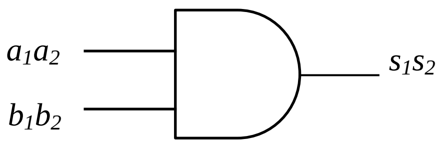


Figure 4.4: AND-gate for 3-value logic

Consider an AND-gate in Figure 4.4 with two inputs a_1a_2 and b_1b_2 and one output s_1s_2. The corresponding truth table is presented in Table 4.2.

a_1a_2	b_1b_2	s_1s_2
00 ('X')	00 ('X')	00 ('X')
10 ('0')	00 ('X')	10 ('0')
00 ('X')	10 ('0')	10 ('0')
10 ('0')	11 ('1')	10 ('0')
11 ('1')	10 ('0')	10 ('0')
10 ('0')	10 ('0')	10 ('0')
11 ('1')	11 ('1')	11 ('1')
11 ('1')	00 ('X')	00 ('X')
00 ('X')	11 ('1')	00 ('X')

Table 4.2: Truth table for AND-gate for 3-value logic

Then a tool for Boolean logic optimization, analysis and synthesis is used to generate the SAT clauses for all primitive gates with subsequent minimization of the number of the clauses in conjunctive normal form. For the AND-gate the expression (4.3) is obtained.

$$(a_2+\neg s_2)(b_2+\neg s_2)(\neg a_2+\neg b_2+s_2)(a_1+\neg b_2+\neg s_1)(b_1+\neg b_2)(a_1+\neg a_2)(\neg a_2+b_1+\neg s_1)$$
$$(a_1+b_1+\neg s_1)(\neg a_1+a_2+s_1)(\neg b_1+b_2+s_1)(s_1+\neg s_2) \tag{4.3}$$

The SAT-based ATPG process starts with computing of SAT clause in CNF for the whole circuit. This SAT-clause is composed of the SAT clauses for all gates in the circuit. After obtaining the SAT-based representation of the circuit, the fault is injected, and fault propagation paths are determined. Then a fault cone, the set of gates affected by fault, is copied. Then the outputs of fault-free cone z and copied faulty cone z_f are compared for the different inputs assignments. If $z \oplus z_f = 1$, or in other words, the compared outputs have different values, then the pattern tests the fault. General ATPG process was described in the subsection 2.1 of chapter 2 in more details.

SAT-based ATPG results in a set of patterns with minimum number of care bits. This is due to minimization algorithm applied in the process of pattern generation. The idea of finding the patterns with minimum number of care bits is based on the bisection search. This algorithm tries to decrease the number of care bits in the pattern till it is possible, and the valid pattern detecting the particular fault is found.

The relation between the circuit size and the SAT representation of the circuit is linear. So, the more gates are in the circuit, the more clauses are in the SAT representation of the circuit. However, the runtime of SAT-based ATPG is not linear. In this subsection we have introduced two- and three-valued ATPGs. For two-valued ATPG for AND-gate, three clauses are needed; in contrast, the three-valued ATPG has eleven clauses for AND-gate representation. Hence, three-valued SAT-based ATPG requires longer computational time and for some circuits it is infeasible.

The decision was made to use commercial ATPG in order to generate test patterns for larger circuits. Commercial ATPG does not require much time. On the other hand, it should be noted that commercial ATPG is a black box tool, so we are not aware of the internal structure and solving methods.

4.2.2 Commercial ATPG

To start off with, we decided to generate test patterns for the whole list of hard-to-detect faults. We aimed at minimization of the number of care bits by introducing “X” values in ATPG process. The commercial tool was very fast in the process of pattern generation. The results are presented in chapter 6.

One of the basic features of commercial ATPG tools is a good compression mechanism. And this is clear since one usually wants to have the smallest set of test patterns generated to save memory. We presume that the commercial tool tends to compress the patterns internally, so it would take advantage of some “don’t care” bits in the case of giving the whole list of faults. Since we aim at the least possible number of care bits, the decision was made to use commercial tool for generation of the patterns for each fault in a fault list, so that commercial tool would not have the possibility to merge the patterns.

This can be achieved by adding one fault at a time from the fault list, running ATPG, saving the current pattern and deleting the fault. This approach is depicted in Figure 4.5. The results are also presented in chapter 6.

Trying to contrast the results obtained by commercial ATPG and SAT-based ATPG, we can take a look at Figure 4.6 and Figure 4.7. The comparison can be made by scrutinizing the number of care bits in the pattern obtained for a particular fault in a fault list.

In Figure 4.6 and Figure 4.7, histograms for the circuit p45k and p100k are depicted. Only the part of the fault list and corresponding patterns are shown. If there was no pattern generated for a particular fault, then there is a blank bar. On the x-axis we have the index of a fault in a fault list, and on the y-axis we have the number of care bits corresponding to the pattern which tests the particular fault designated by an index.

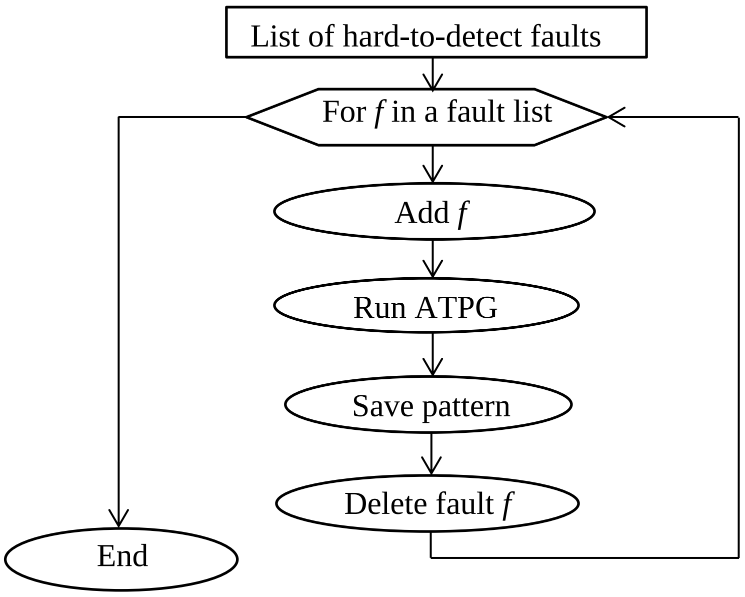


Figure 4.5: Process of generating patterns with commercial ATPG

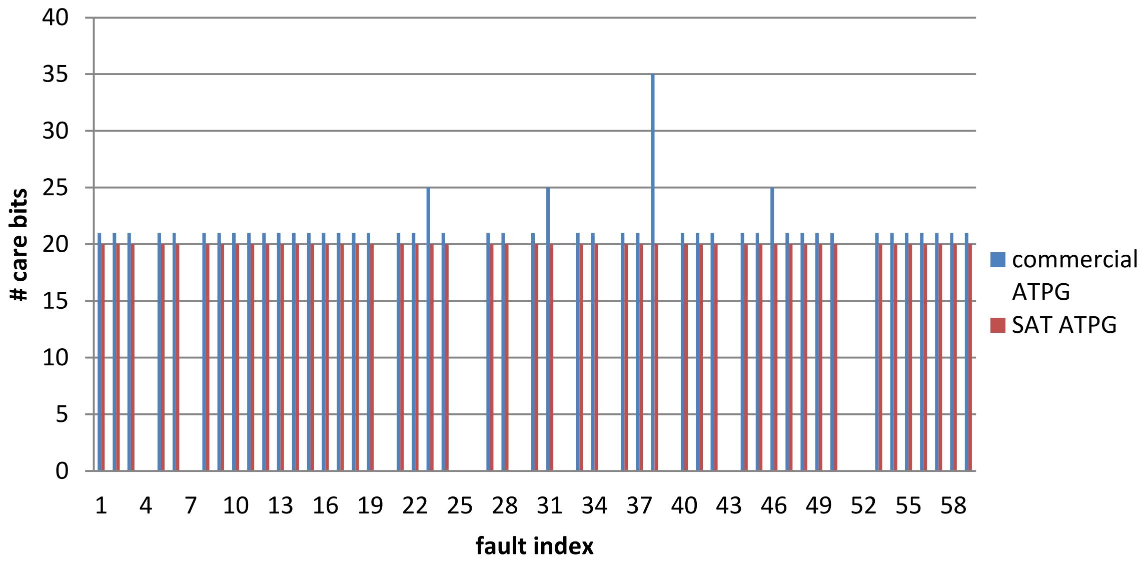


Figure 4.6: Commercial vs. SAT-based ATPG for p45k

For the circuit p45k, SAT-based TPG has generated patterns with less number of care bits. Most of the times the difference in the number of care bits is one. Only in several cases this difference is five or fifteen bits. For the circuit p100k there is the same number of care bits for both TPGs, but sometimes commercial ATPG generated patterns with more care bits.

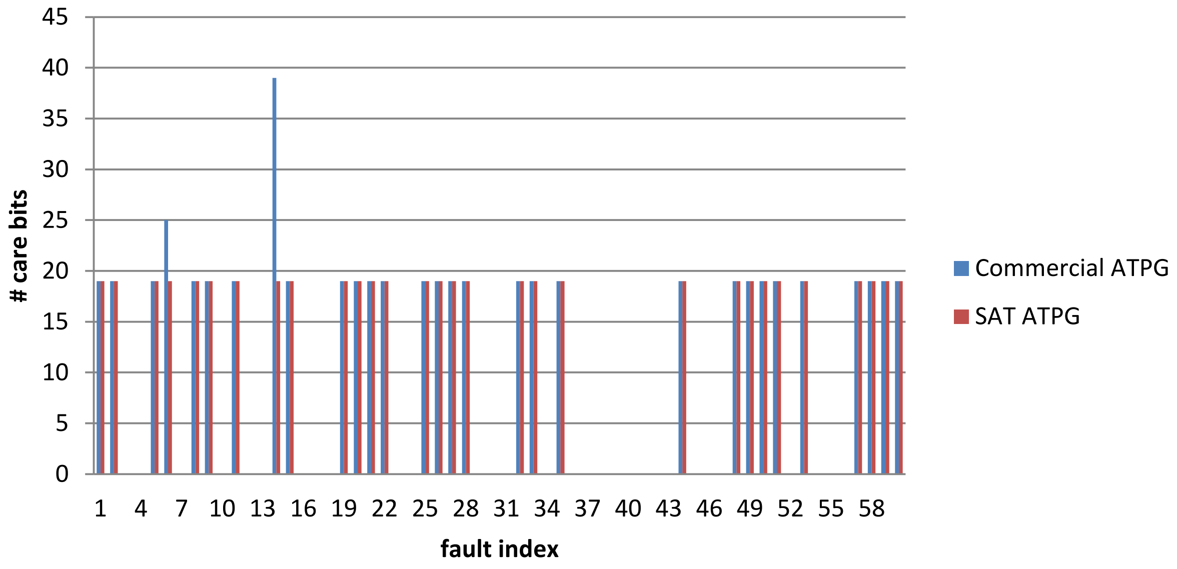


Figure 4.7: Commercial vs. SAT-based ATPG for p100k

Judging by previously shown figures, we conclude that SAT-based ATPG generates patterns with less number of care bits, but it is not applicable to the large circuits. In the next chapter, we are considering the set covering heuristics for choosing polynomials for P-PET taking into account obtained pattern lists.

CHAPTER 5

SET COVERING HEURISTICS

Contents

In this chapter we will concentrate on the algorithm for the selection of feedback polynomials for the test pattern generation of P-PET. In the beginning of the chapter, we will refresh the task formulation and provide the steps required to solve the problem. Then each step will be considered in details.

Let P_{MAX} be the set of primitive polynomials up to degree *MAX*, and K_{MAX} be the set of cones up to the size *MAX*. Let $K_p = \{k \in K_{MAX},\ p \prec k\}$ be the subset of K_{MAX} tested by polynomial p. We also have the pattern list *pl* containing the patterns with minimized number of care bits. The goal for us is formulated as follows: we aim to find the set of primitive polynomials $P_s \subset P_{MAX}$ such that $K_{MAX} = \bigcup_{p \in P_s} K_p$, and the maximized number of covered patterns from *pl*. In other words, we want to encode maximized number of patterns from *pl* into characteristic polynomials which are used for P-PET pattern generation. This goal is subdivided into two sub-goals depending on the requirements. In the first sub-goal, we aim to have the minimum number of polynomials necessary to cover all the cones up to the size *MAX*. This formulation comes from the fact that additional polynomials will cause the test time to rise, and that polynomials need also to be stored on a chip. In this case, the degree of freedom of polynomials choice is exploited to select polynomials which are covering maximized number of patterns. In the second sub-goal, we aim to cover maximized number of patterns, and it is allowed to add extra polynomials. So, in the second stage, the resulting fault coverage is more important.

The procedure of selecting of the primitive polynomials is divided into three steps:

1. Shifting of the cones to the beginning of the scan chain
2. Reduction of the set K_{MAX} by removing redundant cones
3. Iterative selection of polynomials implemented as a set covering heuristics

The first two steps are preparatory. Then in the third step, set covering heuristics is developed in order to solve the problem efficiently.

5.1 Shifting of the cones and removing redundant cones

LFSR with the characteristic polynomial $P(x) = \sum_{i=o}^{n} a_i * x^i$ generates the bit sequence c which can be calculated by (5.1) [9].

$$y_{m+n}=a_0y_m+a_1y_{m+1}+\ldots+a_{n-1}y_{m+n-1},\ m\geq 0 \tag{5.1}$$

For each subsequence $c=c_0,\ c\ ,..,\ c_{n-1}$, there is a sequence $c'=c_{n-1},\ c_0,..,\ c_{n-2}$ where c' is a cyclic shift of c. This means that the position of the cone is unimportant if the relative distances between the cone's inputs are not changed. Consequently, we shift all the cones to the beginning of the scan chain, while keeping the relative distance between their inputs [9].

A cone is considered as **redundant** if all input positions of the cone are contained in other cone. Therefore, we remove all redundant cones.

For example, suppose K_{MAX} contains two cones: $k_1 = \{3, 5, 10\}$ and $k_2 = \{2, 4, 9, 14\}$. Firstly, we shift these cones to the beginning of the scan chain. After shifting, we will get: $k_1 = \{0, 2, 7\}$ and $k_2 = \{0, 2, 7, 10\}$. Cone k_2 contains k_1, or $k_1 \subset k_2$. So, k_1 is redundant cone, and we remove it from K_{MAX}.

5.2 Iterative polynomial selection

In the first sub-goal formulated earlier in this chapter, we aim to maintain the minimum number of polynomials in the final set P_s necessary to cover all the cones up to the size MAX. So, the test time is not increased by addition of extra polynomials. In this task, we use the degree of freedom when selecting polynomials. For example, several polynomials cover the same number of cones from the set K_{MAX}. Then the polynomial which covers the maximum number of care bits is selected.

In the second sub-goal, we are more interested to have further maximized number of the patterns covered by selected polynomials. And we are allowed to increase the number of polynomials in the final set P_s.

In order to control the number of polynomials in the set P_s and the resulting number of patterns covered, we will use a cost function which will assign priorities to the cones and to the patterns. These priorities will be taken into account in the process of polynomials selection.

We will start this subsection with consideration of the general algorithm proposed for the process of polynomial selection. Then we will concentrate on the application of the algorithm for solving both sub-goals. Further the user-defined extension of the algorithm will be provided. And we will conclude this chapter with a summary.

5.2.1 General algorithm

The algorithm proposed is used for achieving both sub-goals. It is flexible since it is configured by means of the coefficients of the cost function. The cost function influences the polynomial selection by considering cones and patterns. To accomplish the first sub-goal, the cost function will give the whole priority to the cones. To reach the second sub-goal, different priorities will be assigned to the cones and to the patterns depending on the user requirements.

Assume that all cones of the circuit are shifted to the beginning of the scan chain, and all redundant cones are removed. So, we have the set of the cones *K* of the circuit.

The algorithm for polynomial selection is presented in Figure 5.1. The first step is to form the set of the cones K_{MAX} (5.2) consisting of the cones from the set *K* with the size up to *MAX*.

$$K_{MAX} = \{k \in K| \; |k| \leq MAX\} \tag{5.2}$$

Then we compose another set of the cones K_d which is comprised of the cones of the size equal to *MAX*. If there are no cones found of the size *MAX*, then we decrease the value of *MAX* by one and continue searching for the cones of decreased *MAX* size. This is reflected in (5.3). In the variable *idx*, we store the size of the cones in the set K_d.

$$1) \; K_d = \{k \in K_{MAX}| \; |k| = MAX\}, idx = MAX \tag{5.3}$$

$$2) \; if \; Kd = 0, MAX = MAX - 1, goto \; step \; 1$$

Given the set of polynomials of degree *idx*, we construct the set of polynomial candidates P_{cnd} according to the cost function which takes into account cones from the set K_d and patterns from the pattern list *pl*. We choose some number of polynomials (*num_polys*) with the largest value of the cost function to be the elements of the set P_{cnd}. This is reflected in (5.4).

$$for \; i = 0 \; to \; num\_polys \; \{$$

$$3) \; P_{cnd} = \{p \in P_{idx}| cost \; function \; is \; maximal\} \tag{5.4}$$

$$4) \; if \; |P_{cnd}| = 0, idx = idx + 1, goto \; 3) \; \}$$

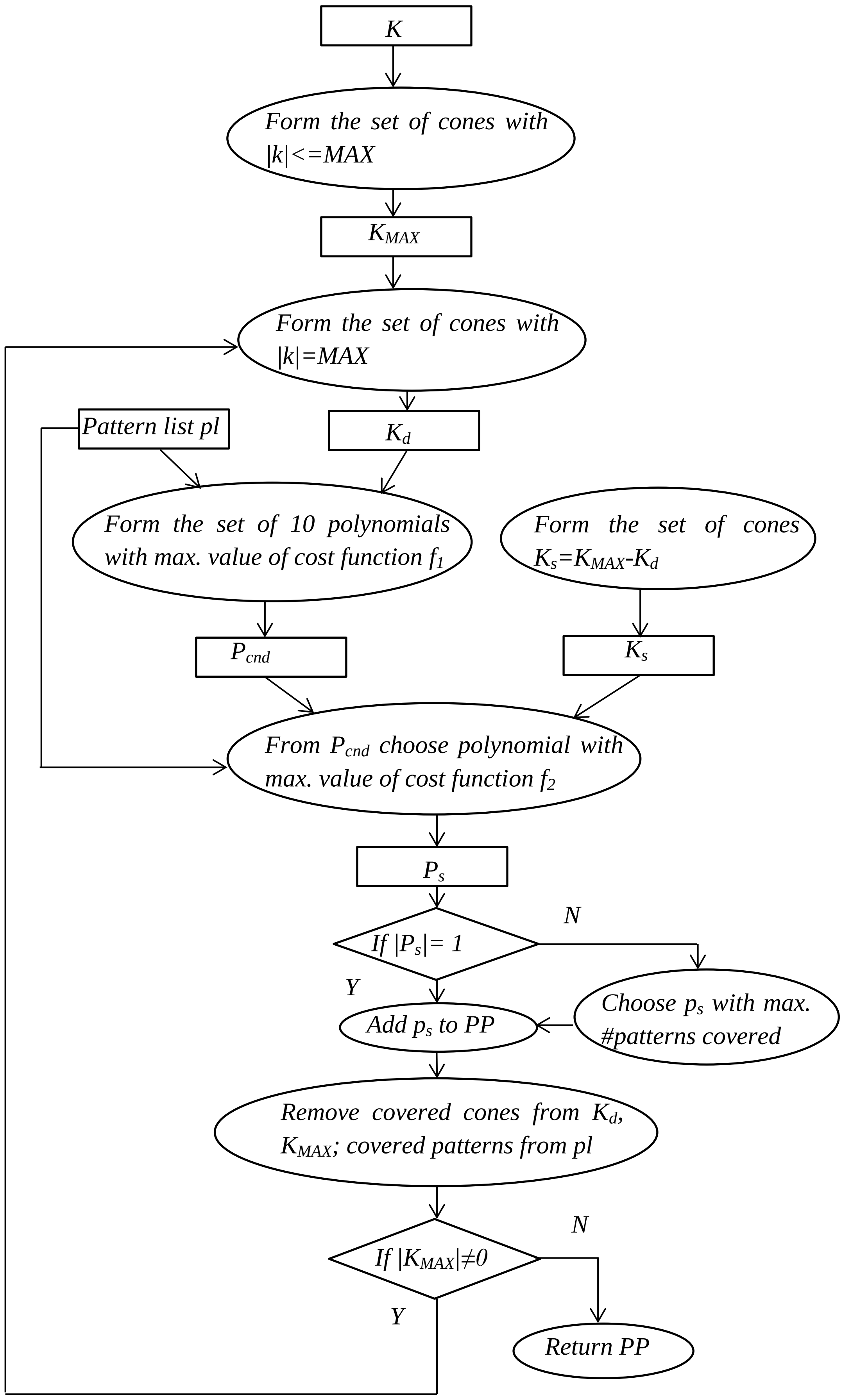

K
Form the set of cones with $|k|<=MAX$
K_{MAX}
Form the set of cones with $|k|=MAX$
Pattern list pl
K_d
Form the set of 10 polynomials with max. value of cost function f_1
Form the set of cones $K_s=K_{MAX}-K_d$
P_{cnd}
K_s
From P_{cnd} choose polynomial with max. value of cost function f_2
P_s
If $|P_s|= 1$
N
Y
Choose p_s with max. #patterns covered
Add p_s to PP
Remove covered cones from K_d, K_{MAX}; covered patterns from pl
If $|K_{MAX}|\neq 0$
N
Y
Return PP

Figure 5.1: Algorithm for the selection of polynomials

The next step in the algorithm is to form the set of the cones K_s which contains the cones from K_{MAX} which are not in K_d (K_s=K_{MAX}-K_d).

$$K_s = \{k \in K_{MAX} | k \notin K_d\} \qquad (5.5)$$

Further on we are choosing from the set of polynomial candidates P_{cnd} those polynomials which have maximal value of the cost function. Cost function takes into account cones from the set K_s and patterns from the pattern list *pl*. In the case that set P_s contains several polynomials, we will choose the one with maximal number of the patterns covered (5.6).

$$P_s = \{p \in P_{cnd} | cost\ function\ is\ maximal\} \qquad (5.6)$$

$$if\ |P_s| \neq 1, choose\ p_s \in P_s\ with\ max. \#\ of\ patterns\ covered$$

Then we add the chosen polynomial p_s to the final set of selected polynomials *PP*. In the next step all covered by the chosen polynomial cones are removed from the sets K_d and K_{MAX}, and all covered patterns are removed from the pattern list *pl*. We check if K_{MAX} is empty. If so, we return the set of selected polynomials *PP*. If not, we continue by returning to the step of forming the set of the cones K_d.

The final set *PP* with primitive polynomials covers all the cones up to the size *MAX* and also the maximized number of the patterns from the pattern list *pl*. Each polynomial from the set *PP* will be used as a feedback polynomial for the LFSR. All possible patterns per polynomial will be applied to the circuit.

In this algorithm, we try to cover the larger cones first when we are selecting polynomial candidates taking into account patterns. Then we target polynomials from the subset of smaller cones. So, the checking whether the polynomial covers the cone $p \prec k$ is applied, firstly, for the cones of the size *MAX* and the set of polynomials of degree *MAX* when we are forming the set of polynomial candidates P_{cnd}. Secondly, the checking is performed for the cones smaller than *MAX* and polynomials from the set P_{cnd}. This approach requires much more less time than checking pairwise $p \prec k$ for all cones of K_{MAX} and the set of polynomials of degree *MAX*.

5.2.1.1 Essential cone handling

There might be a situation where some cones are covered only once by the set of the polynomials of a particular degree. We call that cone *essential*. For instance, the set K_d consists of five cones which are k_1, k_2, k_3, k_4, k_5. Suppose that the corresponding set of polynomials consists of seven polynomials designated by p_1, p_2, p_3, p_4, p_5, p_6, p_7.

In (5.7) the information regarding which cones each polynomial covers is presented. In this example, the cone k_4 is covered only once, so k_4 is an essential cone.

The other cones are covered more than once.

$$\begin{array}{cc} p_1: \{k_1, k_3\} & p_2: \{k_1, k_2, k_5\} \\ p_3: \{k_1, k_3, k_5\} & p_4: \{k_1\} \\ p_5: \{k_2, k_5\} & p_6: \{k_1, k_5\} \\ p_7: \{k_4\} & \end{array} \tag{5.7}$$

The polynomial covering this essential cone is added to the resulting set of polynomials PP anyway. Hence, we extend described algorithm by consideration of the essential cone and adding the polynomial earlier, in the beginning of the heuristics. After forming of the set K_d, we check whether K_d contains the essential cone. If so, we add the polynomial covering this essential cone to the set PP. We also remove from the set of the cones K_d and K_s all covered by the polynomial cones. The covered patterns are eliminated from the pattern list *pl* as well.

5.2.1.2 Realization with minimal number of polynomials

For the first sub-goal, we are selecting the minimal number of polynomials for the set PP. The task is accomplished by using of an appropriate cost function. In the step of selecting the set of polynomial candidates P_{cnd}, the cost function f_1 as in (5.8) is applied. This cost function targeting the number of the cones from the set K_d.

$$f_1 = \#cones\_covered\_in\ \_K_d \tag{5.8}$$

The cost function f_2 (5.9) is used to choose the polynomial from the set P_{cnd} with a maximal number of the cones from the set K_s. If there are several polynomials with maximal value of the cost function f_2, we choose the one with a maximal number of the patterns covered according to the cost function f_3 (5.10).

$$f_2 = \#cones\_covered\_in\_K_s \tag{5.9}$$

$$f_3 = \#patterns\_covered \tag{5.10}$$

This approach ensures that the number of the polynomial in the final set PP is minimal since we are concentrating only on the cones. The number of the patterns covered is considered when we have several polynomials targeting equal maximal number of the cones.

5.2.1.3 Realization with more than minimal number of polynomials

In the second sub-goal, the larger number of the patterns covered is achieved by allowing more than minimal number of polynomials in the final set PP. In this case, during the process of polynomial selection, we consider the cones and the patterns till all cones up to *MAX* size are covered.

The cost function for this realization looks as in (5.11).

$$f = coefficient\_cones * \#cones\_covered + coefficient\_patterns * \#patterns\_covered \tag{5.11}$$

In the cost function (5.11), we have coefficients which influence the outcome in terms of the number of the cones and the patterns covered. These coefficients determine the priorities. When the coefficient for the cones is large than the coefficient for the patterns, the cost function will target more cones than patterns and vice versa. In our algorithm, we use the cost function in two steps: when we are selecting the set of polynomial candidates (f_1) and in the step of the final selection of the polynomial (f_2). In the first case, we are considering the set of the cones K_d and in the second- K_s (5.12).

$$\begin{aligned} f_1 &= coefficient\_cones * \#cones\_covered\_K_d + coefficient\_patterns \\ &\quad * \#patterns\_covered \\ f_2 &= coefficient\_cones * \#cones\_covered\_K_s + coefficient\_patterns \\ &\quad * \#patterns\_covered \end{aligned} \tag{5.12}$$

In the cost function, we can also consider the "weight" or "significance" of the cone. Some cones are covered by the set of polynomials of particular degree more frequently than the other. The more frequently particular cone is covered by the set of polynomials, the less its weight. So, the new cost function looks as in (5.13).

$$f = coefficient\_cones * \textstyle\sum_{i=1}^{n} weight_i + coefficient\_patterns * \#pattern \ , \tag{5.13}$$

where n is the number of the cones covered.

The similar weights calculation can be applied for the patterns as well (5.14).

$$f = coefficient\_cones * \textstyle\sum_{i=1}^{n} weight_i + coefficient\_patterns * \textstyle\sum_{i=1}^{m} weight_i, \tag{5.14}$$

where m is the number of patterns covered.

There are several methods of calculating the weight. One way is to have an inverse of the number of the cones (or patterns) covered by the set of polynomials. So, assume we have in K_d three cones: k_1, k_2, k_3. We start with calculation of the number of the times each cone is covered by the set of polynomials. Suppose, k_1 is covered 2 times, k_2 is covered 10 times and k_3 is covered 5 times. If we invert the number of occurrence, we will get the following weights: 0.5, 0.1 and 0.2 for k_1, k_2 and k_3 respectively.

Another way of calculating the weights will be considered on the following example. Suppose, k_1 is covered 10 times, k_2 is covered 15 times and k_3 is covered 30 times. Firstly, we will assign the weight equal to 1 to the cone with minimal number of occurrences. For the other cones we will use the formula (5.15).

$$weight = 1 - \frac{\#occurence - min}{\#occurence} \ , \tag{5.15}$$

where min is the minimal number of occurrences.

In our example, *min* is equal to 10. The weights of the cones k_2 and k_3 calculated by formula (5.15) are equal to 0.6 and 0.3 respectively. The weights for the patterns are calculated analogously.

While choosing the polynomials, we also will take into account that the cones from the set K_d are more important than the cones from the set K_s. Imagine the situation when all the cones from K_s are already covered, and we have one cone from K_d not covered. Then we add one more polynomial to cover this cone. Concentrating more on the cones from K_d safes us from introducing extra polynomials. This idea is captured in the cost functions f_1 and f_2 presented in (5.16). There f_1 takes into account the number of the covered cones of the set K_d and the number of the covered patterns. In f_2 we are targeting the cones from the set K_d and K_s as well as the patterns, but for the cones from K_s we have additional multiplication by 0.5 to lower the priority of the cones from the set K_s in comparison to the cones from K_d.

$$f_1 = coefficient\_cones \ * \#cones\_covered\_K_d + coefficient\_patterns \ * \#patterns\_covered \tag{5.16}$$

$$f_2 = coefficient\_cones \ * \#cones\_covered\_K_d + 0.5 * coefficient\_cones \ * \#cones\_covered\_K_s + coefficient\_patterns \ * \#patterns\_covered$$

5.3 User-defined introduction of extra polynomial

By means of the cost function we influence the total number of the patterns covered. The increase of the coefficient will induce the increase of the number of the patterns covered, and at the same time more polynomials will be in the final set *PP*. All polynomials from the set *PP* are stored on a chip. And the more polynomials are in the set *PP*, the larger the test time. After covering all the cones in the set K_{MAX}, introduction of additional polynomial should be justified by the number of the patterns it covers. Hence, the algorithm can be extended. The user will define the number of the patterns additional polynomial should cover. This idea is depicted in Figure 5.2.

We have the pattern list *pl* consisting of the patterns not covered by *PP* and the set of polynomials of the certain degree. From this set of polynomials we are choosing the one with maximal number of the patterns covered from *pl*. If the chosen polynomial covers more than the user-defined number *z*, we add the polynomial to the final set of additional polynomials *PP'*. Then we remove from the pattern list *pl* all covered patterns. And continue by searching the polynomial covering the maximal number of the patterns. If the chosen polynomial covers less than *z* patterns, we do not add it to *PP'* and return *PP'*.

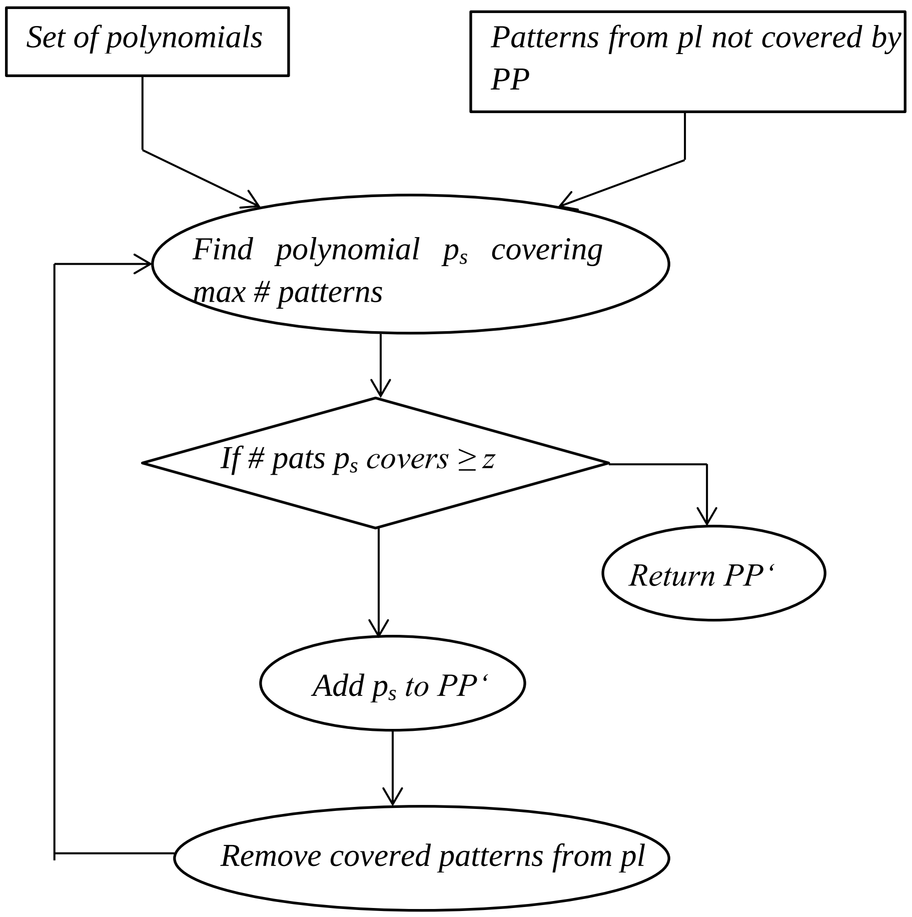

Set of polynomials
Patterns from pl not covered by PP
Find polynomial p_s covering max # patterns
If # pats p_s covers $\geq z$
Return PP'
Add p_s to PP'
Remove covered patterns from pl

Figure 5.2: Algorithm for adding extra patterns

5.4. Summary of the algorithm for polynomial selection

In this subsection we will discuss the algorithm proposed for the polynomial selection. We will concentrate on how to use this algorithm to solve two sub-goals.

In order to show the flexibility of the algorithm, we consider one cost function which is used for achieving both sub-goals. For this purpose, the cost function as in (5.12) or (5.16) is used.

To reach the first sub-goal, we need to apply the cost function with coefficient for the cones equal to 1.0 and coefficient for the patterns equal to 0.0. So, the number of the patterns covered by particular polynomial is not taken into account when forming the set of polynomial candidates P_{cnd}. In the step of choosing the final polynomial p_s, we are not considering the patterns covered, but if there are several candidates with the same number of cones covered, we choose the one with maximal number of patterns covered.

To achieve the second sub-goal, we will use the same algorithm but with different coefficients for the cost function. Both coefficients are not equal to zero. Depending on the coefficients we can prioritize either cones or the patterns covered. The pattern and the fault coverage can be further increased by the extension described in 5.3. There we are adding extra polynomials if they cover at least user-defined number of patterns.

The results of the application of the algorithm for polynomial selection for both sub-goals are presented in chapter 6.

CHAPTER 6

EXPERIMENTAL RESULTS

Contents

In this chapter we will present all experimental results obtained within the thesis. We will start with consideration of the industrial circuits which will be used in further experiments. The results of the fault simulation for obtainment of the list of hard-to-detect faults and test pattern generation will be presented in the subsequent subsections. Then the results of the set covering heuristics for polynomial selection will be discussed. And in the end of the chapter, we will provide the comparison of the proposed approach with usual pseudo-random testing as the first phase of the mixed-mode BIST.

6.1 Circuit characteristics

In this thesis we will use twelve industrial circuits of different sizes. Firstly, we will provide some information about these circuits.

In [9], the distribution of the cones was analyzed to understand which portion of the circuit to test exhaustively. This information helps in the process of choosing the value of *MAX*, the maximum cone size to be applied in P-PET. In [9] it was shown that the majority of the cones are relatively small. There it was proposed to test cones up to the size 24. This gives an optimal trade-off between gate coverage and test time. So, we will further use the value of *MAX* equal to 24. Table 8.1 shows the characteristics of the circuits considered taken from [9]. The first column shows the circuit name; the second reports the number of logic gates; the third column presents the number of primary and pseudo-primary inputs; and in the fourth column we have the number of primary and pseudo-primary outputs.

Primary inputs (PIs) refer to the external inputs to the circuit. Pseudo-primary inputs (PPIs) are scan cell inputs. PIs and PPIs can be set to any logic values. The difference between them is that PIs are set directly from the external inputs, whereas PPIs are set through the scan chain inputs. Primary outputs (POs) are external outputs of the circuit. Pseudo-primary outputs (PPOs) refer to the scan cell outputs. Both POs and PPOs can be observed. The difference is that POs are observed directly from the external outputs, whereas PPOs are observed through the scan chain outputs [46].

The fifth column shows the percentage of the cones up to the size 24, whereas the sixth column presents the percentage of the gates covered by these cones.

Circuit	#gates	#(PI+PPI)	#(PO+PPO)	Cones, %	Gates, %
p35k	46584	2912	2229	74.07	38.54
p45k	45100	3739	2550	57.28	55.26
p89k	90152	4632	4557	64.18	30.54
p100k	96087	5902	5829	82.75	49.76
p141k	174600	11290	10502	45.05	34.54
p239k	261784	18692	18495	83.91	62.41
p259k	336004	18713	18495	83.25	65.84
p279k	293637	18047	17827	58.98	52.16
p378k	374467	15732	17420	68.65	82.54
p418k	442872	30430	29809	58.42	48.04
p483k	510538	33264	32610	85.48	60.08
p533k	652802	33373	32610	83.68	66.66

Table 6.1: Circuits' characteristics [9]

According to the Table 6.1, for the majority of the circuits, more than 60% of the cones are of the size 24 or smaller. For some circuits, this value is more than 80%. The corresponding percentage of the gates covered is in the range from 30% to 82%.

6.2 Fault simulation results

Table 6.2 shows the results of the fault simulation for the circuits: p35k, p45k, p89k, p100k, p141k, p239k, p259k, p279k, p378k, p418k, p483k, p533k. The second column of the Table 6.2 corresponds to the number of the faults from the cones larger than *MAX*. The third column is the number of the faults which were detected by applying one million of pseudo-random patterns. These faults are easy-to-detect. They will be detected by P-PET. The fourth column is the number of faults which were not detected by applying one million of pseudo-random patterns. These faults are hard-to-detect faults which are of interest. They would not be detected by P-PET. We will target these hard-to-detect faults by choosing appropriate feedback polynomials for the P-PET. The last column corresponds to the fault coverage. In this case, the fault coverage shows the percentage of hard-to-detect faults from all faults considered, or faults from the larger than *MAX* cones.

Circuit	**# faults considered**	**# easy-to-detect faults**	**# hard-to-detect faults**	**Fault coverage,%**
p35k	146838	73259	73579	49.891
p45k	106301	105503	798	99.2493
p89k	327086	264586	62500	80.891
p100k	166960	165931	1029	99.38
p141k	287552	276761	10791	96.24
p239k	557946	537714	20232	96.373
p259k	653520	629124	24396	96.266
p279k	723254	659740	63514	91.218
p378k	414050	414050	0	100
p418k	1139182	1045919	93263	91.813
p483k	1115568	1089318	26250	97.6469
p533k	1270612	1242006	28606	97.748

Table 6.2: Results of fault simulation

The column with hard-to-detected faults is of more importance to us. The more faults in this column, the more corresponding patterns would need to be generated in the second phase of the mixed-mode BIST. The percentage of hard-to-detect faults varies from 50% to 99.2%.

For the circuit p378k, we got complete fault coverage of 100%. It means that all the faults from the larger than *MAX* cones of p378k are tested during P-PET. For the other circuits, we need to generate deterministic patterns for obtained list of hard-to-detect faults and further use them in the process of selection of feedback polynomials for P-PET.

6.3 Results of test pattern generation

The results of test pattern generation for hard-to-detect faults are presented in Table 6.3. In this table, we combined the results of three approaches for ATPG discussed in chapter 4. The first column is the circuit name. The second column (“# htd. faults”) shows the number of hard-to-detect faults. The three following group the results for SAT-based ATPG, commercial ATPG for the whole list of faults (“Commercial(1)”) and commercial ATPG for each fault in a fault list (“Commercial(2)”).

Consider SAT-based ATPG results. According to the table, test patterns were generated only for three circuits (p45k, p100k, p141k) since the other circuits were too large or had too many hard-to-detect faults. Table 6.3 shows for each ATPG approach the number of patterns generated (# pats), number of the faults detectable by these patterns (#det. faults) and corresponding fault coverage. Since the circuit contains the faults for which it is not possible to generate patterns, we have not complete fault coverage. These faults are untestable; they manifest redundancy in the circuit. Table 6.3 can also demonstrate the quality of TPG process. We can compare the fault coverage for a SAT-based ATPG and a commercial tool. For p45k, p100k the fault coverage is almost the same, whereas for p141k the fault coverage of SAT ATPG is higher. Fault coverage of two applications of commercial tool is the same, since one ATPG tool was used. Fault coverage varies for different circuits from 29% till 88%. The value of fault coverage and corresponding number of detectable faults is the upper bound for us. We will not be able to cover more faults than the number of faults detectable by the patterns generated.

circuit	# htd. faults	SAT ATPG			Commercial (1)		Commercial(2)		
		# pats	#det. faults	FC, %	# pats	#det. faults	# pats	#det. faults	FC, %
p35k	73579	not performed			6586	56944	15325	56944	77.4
p45k	798	544	597	74.8	67	597	129	597	74.8
p89k	62500	not performed			7049	45563	13100	45563	72.9
p100k	1029	465	501	48.6	206	500	344	500	48.6
p141k	10791	7580	8448	78.2	4677	7496	5996	7496	69.46
p239k	20232	not performed			2237	14144	2653	14144	69.9
p259k	24396	not performed			2952	17597	3537	17597	72.13
p279k	63514	not performed			5844	40177	8672	40177	63.25
p418k	93263	not performed			13242	82661	18379	82661	88.63
p483k	26250	not performed			1460	7702	1944	7702	29.34
p533k	28606	not performed			1911	9474	2570	9474	33.12

Table 6.3: ATPG results

Not only the fault coverage is of interest for us, but also the number of care bits each pattern contains. The less number of care bits the pattern has, the greater the probability to cover the pattern. Table 6.4 shows the number of patterns generated with particular number of care bits for each circuit. There the results for three ATPG approaches are presented. There are several ranges of numbers of care bits: less than 30, from 30 to 40, from 40 to 50, and the last range contains the patterns with more than 50 care bits. We are interested in the range up to 40 care bits, because of the greater possibility to find the feedback polynomial of maximal degree 24 covering the pattern. Talking about SAT-based ATPG, for p45k we have 43% of the patterns with up to 40 care bits, it is equal to 233 patterns; for p100k we have 68% of patterns up to 40 care bits, that is 316 patterns; and for p141k we have 30% of patterns with up to 40 care bits, it is equal to 2274 patterns. The results of running commercial ATPG for each fault are comparable with the results obtained by running commercial ATPG for the whole list of faults. For p35k, there was no pattern generated with less than 40 care bits. So, it is not possible to cover any pattern for p35k. For the other circuits, the results are more optimistic. Summing up, the percentage of patterns with the number of care bits up to 40 varies from 26% to 76%.

circuit	**SAT ATPG #pats with #cb**				**Commercial (1) #pats with #cb**				**Commercial (2) #pats with #cb**			
	< 30	**30-40**	**40-50**	> 50	< 30	**30-40**	**40-50**	> 50	< 30	**30-40**	**40-50**	> 50
p35k	not performed				0	0	0	100	0	0	0	100
p45k	5	38	40	17	14	13	22	51	12	30	16	42
p89k	not performed				32	13	14	41	19	14	11	56
p100k	29	39	20	12	36	17	23	24	33	17	24	26
p141k	22	8	3	67	40	20	5	35	35	18	5	42
p239k	not performed				23	37	31	9	25	35	31	9
p259k	not performed				32	23	37	8	33	21	36	10
p279k	not performed				14	18	12	16	10	16	12	62
p418k	not performed				37	23	15	25	33	23	15	29
p483k	not performed				31	36	10	23	36	35	10	19
p533k	not performed				33	39	9	19	38	38	8	16

Table 6.4: Distribution of the patterns with different number of care bits

6.4 Set covering heuristics results

The results of the set covering heuristics will be considered in terms of the number of polynomials in the final set *PP*, the number of patterns covered and corresponding number of faults covered. According to the algorithm presented in chapter 5, the number of resulting polynomials and covered patterns depends on the values assigned to the coefficients of the cost function.

To start off with, we will consider the heuristics with *coefficient_cones* equal to 1.0 and *coefficient_patterns* equal to 0.0. This realization is to reach the first sub-goal. In this case, the number of resulting polynomials is minimal. So, this is a lower bound for us in terms of the number of polynomials and number of patterns and faults covered.

In Table 6.5 the results for the described realization are presented. The first column is the circuit name. The asterisk symbol (*) in the circuit name means that the pattern list generated by SAT-based ATPG is used, other realizations are for the pattern lists generated by commercial ATPG (for each fault in a fault list). The second column shows the number of detectable hard-to-detect faults. The third column is the resulting number of polynomials of the set *PP* with corresponding degree of polynomial. For example, for p35k we have $1*2^{16}+2*2^{11}$. This means that in the set *PP* there are two polynomials: one of degree 16 and another of degree 11. The forth column is the number of covered by the polynomials from the set *PP* patterns. The fifth column is the corresponding number of covered by the patterns hard-to-detect faults. And the last column shows the percentage of faults covered from detectable hard-to-detect faults.

According to the Table 6.5, the percentage of covered faults varies from 5% to 29.5%. So, for some circuits a good percentage of the faults covered is achieved without introducing extra polynomials. The results obtained for the pattern lists of commercial and SAT-based ATPGs are comparable.

circuit	**#detectable hard-to-detect faults**	**Req. polys**	**#patterns covered**	**#faults covered**	**%of faults covered**
p35k	56944	$1*2^{16}+2*2^{11}$	0	0	0
p45k	597	$1*2^{24}$	6	32	5.36
p45k*	597	$1*2^{24}$	6	30	5.02
p89k	45563	$1*2^{24}+1*2^{23}$	2106	7397	16.23
p100k	500	$2*2^{24}$	91	136	27.2
p100k*	501	$2*2^{24}$	112	148	29.54
p141k	7496	$2*2^{24}+2*2^{23}$	819	1087	14.5
p141k*	8448	$2*2^{24}+1*2^{23}$	861	1100	13.02
p239k	14144	$3*2^{24}+1*2^{23}$	547	2726	19.27
p259k	17597	$4*2^{24}$	886	4270	24.26
p279k	40177	$3*2^{24}$	587	3418	8.5
p418k	82661	$5*2^{24}$	3322	15298	18.5
p483k	7702	$5*2^{24}+1*2^{23}$	435	1217	15.8
p533k	9474	$8*2^{24}$	718	2132	22.5

Table 6.5: Heuristics results for *coefficient_cones*=1.0, *coefficient_patterns*=0.0

There is a possibility to further improve the percentage of covered hard-to-detect faults as it was requested in the second sub-goal by means of introducing *coefficient_patterns* larger than 0.0. In this case, the number of polynomials required might be increased depending on the coefficients. Table 6.6 demonstrates the results of running the heuristics for different pairs of coefficients. The cost function here is based on the number of cones and patterns covered (5.12). The first column of the Table 6.6 shows the circuit name. As in the Table 6.5, the asterisk symbol (*) corresponds to the realization with exact solution for the test pattern generation. The second column is the number of detectable hard-to-detect faults. The third and the forth columns are values for *coefficient_cones* and *coefficient_patterns*. The fifth column shows the number of polynomials in the set *PP*. These polynomials are all of degree 24. The sixth and the seventh columns are the number of patterns and faults covered. And the last column is the percentage of the faults covered from detectable hard-to-detect faults. We will refer to this percentage further as to the fault coverage. The percentage of covered faults varies from 5% to 45%. For some circuits introducing of additional polynomials causes the dramatic increase in the number of the faults covered. We will consider results for some circuits in details.

For the circuit p45k, the minimum number of polynomials to cover all the cones up to *MAX* size is one. And according to the Table 6.5, the percentage of covered faults is 5.36%. In Table 6.6 the same results are presented for two pairs of coefficients. If we increase the value of *coefficient_patterns* up to 0.5 or 0.7, we can reach the fault coverage of 25.79% and 26.80% correspondingly by adding two extra polynomials to the final set *PP*. The results for the pattern list generated by exact solution ATPG are comparable with the results obtained for the pattern list of obtained by commercial ATPG. Interestingly, for the circuit p89k any pair of coefficients results in the same outcome. Three extra polynomials give increase in the fault coverage of 6.02%.

circuit	**#detectable htd. faults**	***coeff_ con***	***coeff_ patt***	**#polys**	**#pat. covered**	**#faults covered**	**%of faults covered**
p45k	597	0.1	0.9	5	32	187	31.32
		0.3	0.7	3	23	160	26.80
		0.5	0.5	3	24	154	25.79
		0.7	0.3	1	6	32	5.36
		0.9	0.1	1	6	32	5.36
p45k*	597	0.1	0.9	4	33	153	25.63
		0.3	0.7	4	31	153	25.63
		0.5	0.5	3	18	137	22.95
		0.7	0.3	3	18	137	22.95
		0.9	0.1	1	6	30	5.02
p89k	45563	0.1	0.9	5	2582	10139	22.25
		0.3	0.7	5	2582	10139	22.25
		0.5	0.5	5	2582	10139	22.25
		0.7	0.3	5	2582	10139	22.25
		0.9	0.1	5	2582	10139	22.25
p100k	500	0.1	0.9	7	139	229	45.80
		0.3	0.7	4	125	204	40.80
		0.5	0.5	4	125	204	40.80
		0.7	0.3	3	112	166	33.20
		0.9	0.1	3	110	163	32.60
p100k*	501	0.1	0.9	6	146	185	36.93
		0.3	0.7	4	139	175	34.93
		0.5	0.5	3	135	176	35.13
		0.7	0.3	3	135	176	35.13
		0.9	0.1	2	115	142	28.34
p141k	7496	0.1	0.9	11	2073	2570	34.28
		0.3	0.7	9	1930	2413	32.19
		0.5	0.5	9	1933	2437	32.51
		0.7	0.3	7	1778	2246	29.96
		0.9	0.1	7	1754	2221	29.63

circuit	#detectable htd. faults	*coeff_ con*	*coeff_ patt*	#polys	#pat. covered	#faults covered	%of faults covered
p141k*	8448	0.1	0.9	10	1549	1908	22.58
		0.3	0.7	8	1451	1835	21.72
		0.5	0.5	8	1450	1835	21.72
		0.7	0.3	8	1454	1808	21.40
		0.9	0.1	5	1187	1504	17.80
p239k	14144	0.1	0.9	9	691	3396	24.01
		0.3	0.7	10	700	3365	23.79
		0.5	0.5	7	656	3098	21.90
		0.7	0.3	5	613	3105	21.95
		0.9	0.1	4	577	2788	19.71
p259k	17597	0.1	0.9	10	1163	5475	31.11
		0.3	0.7	10	1163	5475	31.11
		0.5	0.5	9	1132	5407	30.73
		0.7	0.3	6	1030	4799	27.27
		0.9	0.1	4	911	4716	26.80
p279k	40177	0.1	0.9	11	1061	5554	13.82
		0.3	0.7	10	1034	5051	12.57
		0.5	0.5	10	1036	5215	12.98
		0.7	0.3	7	926	4741	11.80
		0.9	0.1	4	744	3997	9.94
p418k	82661	0.1	0.9	15	6554	27591	33.37
		0.3	0.7	15	6554	27591	33.37
		0.5	0.5	15	6530	27043	32.71
		0.7	0.3	14	6398	26581	32.15
		0.9	0.1	10	5734	24743	29.93
p483k	7702	0.1	0.9	13	798	2287	29.69
		0.3	0.7	11	753	2282	29.62
		0.5	0.5	10	719	2480	32.19
		0.7	0.3	8	651	1888	24.5
		0.9	0.1	7	591	1780	23.12
p533k	9474	0.1	0.9	18	1097	3383	35.71
		0.3	0.7	15	1056	3223	34.1
		0.5	0.5	13	988	3008	31.75
		0.7	0.3	10	882	2682	28.3
		0.9	0.1	8	754	2190	23.11

Table 6.6: Heuristics results for different coefficients

For p100k, adding one extra polynomial causes the rise in the fault coverage of 6%. If we add two more polynomials, we will receive the fault coverage of 40.80%. Applying seven polynomials results in covering 45.8% of detectable hard-to-detect faults.

For the circuit p483k, introducing of one additional polynomial causes the increase of fault coverage of 7.32%. To reach the fault coverage of 30% extra seven polynomials are added. For the circuit p279k, one additional polynomial causes the fault coverage to rise up to 1.5%. The fault coverage of 12.57% is reached by means of introducing seven more polynomials. Overall, we conclude that the coefficients regulate the resulting number of patterns covered and the number of polynomials. On the example of p279k, we note that sometimes change in the coefficients results in a not completely predicted outcome. If *coefficient_cones* is equal to 0.3 and *coefficient_patterns* is equal to 0.7, then we have totally 1034 patterns covered. If the corresponding coefficients are 0.5 and 0.5, then 1036 patterns covered even so in the first case we gave more priority to the patterns than in the second case.

For p533k, we obtained the fault coverage of 23.11% without adding extra polynomial. If we add two polynomials, we can reach the fault coverage of 28.3%. Adding of extra eight polynomials gives us the fault coverage of 35.71%.

The number of faults each patterns covers is not always the same, some patterns cover more faults than the other. There are cases when we are covering less number of patterns but the total fault coverage is higher. Consider the results of heuristics for the circuit p483k. Assigning the coefficient for the cones to 0.3 and coefficient for the patterns to 0.7, we covered 753 patterns and 2282 faults. When both coefficients are 0.5, we got 719 patterns covered and 2480 faults. This is a good example showing that less number of patterns can cover more faults.

According to the Table 6.6, the maximum percentage of the faults covered is around 45%. It is worth noting that for each circuit there are number of patterns which can be covered by the polynomials of maximal degree 24. This depends on the number of care bits each pattern has. The distribution of the patterns with different numbers of care bits in the Table 6.4 gives a hint on the number of patterns which can be covered for each circuit. For example, for the circuit p483k, the percentage of the patterns with up to 40 care bits is 71%. By applying 13 polynomials, we are covering 41% of the patterns. So, 30% more of the patterns can be covered by means of adding extra polynomials.

As discussed in chapter 5, we can use a more complex cost function taking into account the weights of the cones as in (5.8). We will present the results of the heuristics with new cost function in Table 6.7. In this realization, weights are calculated by the formula (5.15).

According to the Table 6.7, the results of heuristics are comparable to the results of heuristics without weights consideration. For p483k, the maximum percentage of the faults covered is 30.58% and for p533k - 38.72%.

Up to now, we were discussing results of the heuristics based on the cost functions for polynomial selection. In this algorithm, we stop the iterative polynomial selection as soon as all cones up to the size *MAX* are covered.

The next realization extends the heuristics. After covering all cones we target only patterns in the cost function. We add the polynomial to the final set *PP* if the number of patterns it covers is more or equal than the user-defined number of patterns *num_pat*

(Figure 5.2).

circuit	#detectable hdt. faults	*coeff_ con*	*coeff_ patt*	#polys	#pat. covered	#faults covered	%of faults covered
p483k	7702	0.1	0.9	14	816	2356	30.58
		0.3	0.7	13	795	2267	29.43
		0.5	0.5	10	734	2314	30.04
		0.7	0.3	8	657	2073	26.91
		0.9	0.1	7	591	1780	23.11
p533k	9474	0.1	0.9	22	1167	3408	35.97
		0.3	0.7	21	1148	3669	38.72
		0.5	0.5	19	1112	3485	36.78
		0.7	0.3	14	998	2975	31.4
		0.9	0.1	9	808	2433	25.68

Table 6.7: Results of heuristics with weights consideration

The Table 6.8 shows the results of this extension of the heuristics.

	circuit	#det. htd. faults	*coeff_ con*	*coeff_ patt*	*num_ pat*	#polys	#pat	#faults	%of faults
without weights	p418k	82661	0.9	0.1	50	29	7632	31816	38.48
	p418k	82661	0.7	0.3	50	28	7561	31359	37.93
	p418k	82661	0.5	0.5	50	28	7568	31426	38.02
	p483k	7702	0.9	0.1	10	25	913	3208	41.65
	p533k	9474	0.9	0.1	10	31	1223	3508	37.02
with weights	p483k	7702	0.9	0.1	10	25	913	3208	41.65
	p483k	7702	0.7	0.3	10	25	932	3277	42.54
	p533k	9474	0.9	0.1	10	33	1238	3685	38.89

Table 6.8: Results of heuristics with user-defined polynomial introduction

The first column differentiates between two realizations: without weights consideration (as in Table 6.6) and with weights calculation (as in Table 6.7). The second column is the circuit name. The next column is the number of detectable hard-to-detect faults ("#det. htd. faults"). The following two columns are the coefficient for the cones and the coefficient for the patterns. The sixth column is the least number of patterns which every additional polynomial should cover. The last four columns are the number of polynomials in the final set, the number of patterns covered, the number of faults covered and the percentage of the faults covered from detectable hard-to-detect faults.

For the circuit p418k, addition of extra 19 polynomials of degree 24 gives increase in the fault coverage of 8.55%. For p483k, extra 18 polynomials result in 19.42% increase of fault coverage. And for p533k, additional 23 polynomials cause the rise of fault coverage in 12.71%.

Certainly, given different values for *num_pat*, the resulting number of faults covered and polynomials in the final set will be different, but still introduction of every additional polynomial is determined by the user.

6.5 Comparison of P-PET and pseudo-random testing

In this subsection, we compare the optimized P-PET with usual pseudo-random testing (PRT). We perform fault simulation, and for both approaches we apply the same number of the test patterns. For P-PET we use the realization with minimal number of polynomials in the final set (Table 6.5). We perform fault simulation for the following circuits: p35k, p45k, p89k and p100k.

circuit	**#patterns**	**#faults**	**#undetected faults by PRT**	**#undetected faults by P-PET**	**Difference, %**
p35k	72544	54434	16430	15604	5.02
p45k	16780956	60509	159	112	29.55
p89k	25170442	134478	8798	8138	7.5
p100k	33560335	144159	443	355	19.86
p100k	50337551	144149	439	340	22.55

Table 6.9: Single stuck-at fault coverage

In Table 6.9 the results of the fault simulation for pseudo-random testing and P-PET are presented. The fault model used in the fault simulation is single stuck-at fault. The first column is the circuit's name. The next column shows the number of patterns applied in the fault simulation. The column "#faults" shows the total number of stuck-at-faults considered. The forth and the fifth columns present the number of faults undetected by pseudo-random testing and by the P-PET correspondingly. And the last column shows the difference between the two approaches in percentage, or the percentage of faults P-PET covers more compared to PRT. This difference is calculated by the formula (6.1).

$$\frac{\#undetected\_faults\_PRT - \#undetected_{faults_P} - PET}{\#undetected\_faults\_PRT} \qquad (6.1)$$

For all the circuits, P-PET has higher fault coverage than pseudo-random testing.

For the circuit p100k, we performed fault simulation twice. For the first time, we generated patterns for the fault simulation using two polynomials. This was the realization with minimal number of polynomials. As a result, P-PET has not covered 355 faults, and PRT has not detected 443 faults. The advantage of the P-PET in percentage of the faults covered more is 19.86%. For the second time, we used the polynomial set for the circuit p100k with one extra polynomial. This set was generated as a solution for the second sub-goal. To be more precise, this is the realization for p100k with *coefficient _cones* equal to 0.7 and *coefficient_patterns* equal to 0.3. This time P-PET covered 15 faults more. And the difference with PRT rose up to 22.55%. According to the Table 6.5 and the Table 6.6, the difference between two considered realizations of P-PET for p100k is in 18 faults. In these tables, we were considering the percentage of the faults covered from hard-to-detect faults. And in subsection 4.1 we noted that the set of hard-to-detect faults is a superset of the real faults not detected by P-PET. This explains why we the actual difference in the number of the faults detected is less.

Table 6.9 shows that P-PET with embedded deterministic patterns behaves better than pseudo-random test. The realization of P-PET when we are targeting more faults by means of adding extra polynomials results in increasing of the percentage of the faults covered. In the case of pseudo-random testing, the increase in the number of the patterns does not give a remarkable benefit since fault coverage is eventually saturating. The random-pattern resistant faults cannot be detected by pseudo-random patterns. From this perspective, P-PET with embedded deterministic patterns is a more attractive solution.

6.6 Summary and discussions

The algorithm proposed in this chapter for the selection of characteristic polynomials is very flexible and it can be adjusted depending on the needs of the user. It is used to reach both the sub-goals formulated in the chapter 3. The flexibility is achieved by changing the coefficients of the cost function. In addition, the user can define the minimal number of patterns each additional polynomial should cover.

The results presented in this chapter show that a good percentage of the faults can be covered in a P-PET. If the user is interested in the minimal number of polynomials in the final set, then the percentage of the faults covered from hard-to-detect faults varies from 5% to 30%. This percentage can rise up to 45%, when the possibility is given to increase the number of polynomials in the final set. The percentage is further increased by introducing additional polynomials targeting only patterns.

There is a limit of possible number of the patterns which can be covered for each circuit. This limit comes from the care bit distributions of the patterns. Patterns with more than 40 care bits are unlikely to be covered by the polynomials of degree 24.

Consider the circuit p533k. The care bit distribution diagram is shown in Figure 6.1. According to the Figure 6.1, the percentage of patterns with up to 40 care bits is 76. Since the total number of patterns is 2570, the number of patterns with up to 40 care bits is 1954. So, we can cover maximum 1954 number of patterns for p533k, but the polynomial count will be very high. According to the Table 6.8, we are covering 1238 patterns by 33 polynomials, and each extra polynomial should cover at least 10 patterns. If we would like to continue adding polynomials to increase the coverage, we should consider polynomials which are covering less than 10 patterns.

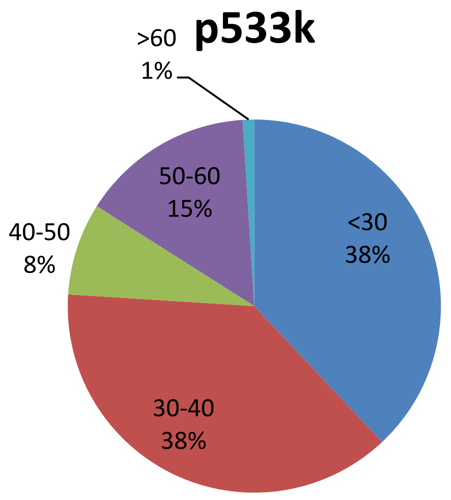


Figure 6.1: Care bit distribution diagram for p533k

In this chapter we also conducted the comparison of optimized P-PET and pseudo-random testing by means of the fault simulation. The results show the advantages of P-PET in terms of the fault coverage.

CHAPTER 7

CONCLUSIONS

In this thesis, several state-of-the-art approaches for mixed-mode BIST scheme were researched and evaluated. As the basis, contemporary P-PET in the first phase of mixed-mode BIST was considered. The second phase is generation of deterministic test patterns for the faults not detected by the first phase. P-PET gives higher defect coverage in comparison with usual pseudo-random testing, but the number of the faults not detected may still be large to encode and store them as deterministic patterns.

The need and the possibility for an improvement of the scheme were observed. And the idea of optimizing P-PET with respect to the fault coverage was introduced. This idea is based on embedding deterministic test patterns from the second phase of mixed-mode BIST into P-PET by appropriate selection of characteristic polynomials used for P-PET pattern generation.

The algorithm which selects the set of characteristic polynomials for P-PET test pattern generation was extended to consider and encode deterministic test patterns. The developed algorithm is very flexible in terms of the resulting number of the pattern count, covered faults as well as in terms of the amount of chosen polynomials.

The proposed algorithm was applied to state-of-the-art industrial circuits, and positive results were obtained. Using the same number of polynomials as in P-PET, the amount of undetected faults could be reduced by 5% to 29.5% compared to P-PET. If more polynomials are allowed, this reduction can be further improved. The upper bound for the percentage of the additional faults covered is determined by the care bits distribution of the patterns to be encoded in P-PET.

Overall, the use of the approach results in an enhanced defect coverage and a lower number of deterministic patterns needed in the second phase to reach a certain fault coverage. The test costs and test application time are comparable with other techniques for mixed-mode BIST. Optimized P-PET is especially appealing when the size of the pattern set needed for the second stage of a mixed-mode BIST scheme is of concern.

The comparison of the proposed technique to pseudo-random testing proves the advantage in terms of fault coverage.

In the thesis, it was pointed out that the resulting fault coverage can be improved by considering the number of faults each deterministic test pattern covers in the process of encoding. This way, not only patterns, but faults are targeted. It is also worth to use another fault models when selecting the set of the faults of interest. In the thesis, only faults of stuck-at model were considered.

The proposed extension of P-PET can be applied in the following manner. Firstly, P-PET with minimum number of polynomials is applied. Secondly, for undetected faults the test patterns are generated. Then the implemented algorithm selects polynomials for P-PET taking into account these patterns. And, lastly, the P-PET with chosen polynomials is performed. In this way, the defect coverage will be higher since we are targeting genuine set of the patterns during P-PET.

In conclusion, a significant improvement in terms of defect coverage was reached. The results show the effectiveness of P-PET when deterministic test patterns are considered during the polynomial selection.

REFERENCES

[1] A.-W. Hakmi, H. J. Wunderlich, C. G. Zoellin, A. Glowatz, F. Hapke, J. Schloeffel, L. Souef, ‘‘Programmable deterministic built-in self-test’’, in Proceedings International Test Conference, 2007, pp. 1-9.

[2] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, B. Courtois, ‘‘Built-In test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers’’, IEEE Transactions on computers, Vol. 44, NO. 2, 1995, pp. 223-232.

[3] A. Jas, J. Ghosh-Dastidar, M. Ng, N. A. Touba, ‘‘An efficient test vector compression scheme using selective Huffman coding’’, IEEE Transactions on Computer-Aided Design, 22(6), 2003, pp. 797-806.

[4] B. Koenemann, ‘‘LFSR-coded test patterns for scan designs’’, in Proceedings European Test Conference, 1991, pp. 27-34.

[5] S. M. Reddy, K. Miyase, S. Kajihara, I. Pomeranz, ‘‘On test data volume reduction for multiple scan chain designs’’, in Proceedings IEEE VLSI Test Symposium, 2002, pp. 103-108.

[6] L. T. Wang, C. W. Wu, X. Wen, ‘‘VLSI test principles and architectures: design for testability’’, Morgan Kaufmann Pub, 2006, pp. 308-310, 344-350.

[7] S. Hellebrand, S. Tarnick, B. Courtois, J. Rajski, ‘‘Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers’’, in Proceedings International Test Conference, 1992, pp. 120-129.

[8] A. Mohammad, “Deterministic BIST”, ELE 6306 Project, Ecole Polytechnique, Montreal, 2004, pp. 1-7.

[9] A. Mumtaz, M. E. Imhof, H. J. Wunderlich, ‘‘P-PET: partial pseudo-exhaustive test for high defect coverage’’, in Proceedings International Test Conference (ITC), 2011, pp. 1-8.

[10] P. H. Bardell, W. H. McAnney, J. Savir, ‘‘Built-In Test for VLSI: pseudo-random techniques’’, John Wiley & Sons, Somerset, NJ, 1987.

[11] E. J. McCluskey, ‘‘Logic design principles: with emphasis on testable semi-conductor circuits’’, Prentice Hall, Englewood Cliffs, NJ, 1986.

[12] M. Renovell, P. Huc, Y. Bertrand, ‘‘CMOS bridging fault modeling’’, in Proceedings IEEE VLSI Test Symposium, 1994, pp. 392-397.

[13] M. Abramovici, M. A. Breuer, A. D. Friedman “Digital system testing and testable design”, The Institute of Electrical and Electronics Engineering, New York, 1990, pp. 343-412, 457-524.

[14] H. J. Wunderlich, S. Hellebrand, “Generating pattern sequences for the pseudo-exhaustive test of MOS-circuits”, Fault-Tolerant Computing, FTCS-18, Digest of Papers, in Proceedings IEEE Eighteenth International Symposium, 1988, pp. 36-41.

[15] E. J. McCluskey, “Verification testing—A pseudoexhaustive test technique”, Computers, in Proceedings IEEE Transactions on 100.6, 1984, pp. 541-546.

[16] N. Devtaprasanna, A. Gunda, P. Krishnamurthy, S. M. Reddy, I. Pomeranz, “A unified method to detect transistor stuck-open faults and transition delay faults”, in Proceedings IEEE Test Symposium, 2006, pp.185, 192.

[17] S. Hellebrand, H. J. Wunderlich, “Tools and devices supporting the pseudo-exhaustive test”, in Proceedings conference on European design automation, IEEE Computer Society Press, 1990, pp. 13-17.

[18] E. J. McCluskey, C. W. Tseng, “Stuck-fault tests vs. actual defects”, in Proceedings International Test Conference, 2000, pp. 336-342.

[19] C. W. Tseng, S. Mitra, S. Davidson, E. J. McCluskey, “An evaluation of pseudo random testing for detecting real defects”, in Proceedings IEEE VLSI Test Symposium, 2001, pp. 404-409.

[20] A. A. Ismaeel, “Testing for stuck faults in CMOS combinational circuits”, Circuits, Devices and Systems, IEE Proceedings G, vol.138, no.2, 1991, pp.191, 197.

[21] D. T. Tang, C.-L. Chen, “Logic test pattern generation using linear codes”, Computers, IEEE Transactions on 100.9, 1984, pp. 845-850.

[22] L.-T. Wang, E. J. McCluskey, “Condensed linear feedback shift register (LFSR) testing—a pseudo-exhaustive test technique”, Computers, IEEE Transactions on 100.4, 1986, pp. 367-370.

[23] C. L. Chen, “Exhaustive test pattern generation using cyclic codes”, Computers, IEEE Transactions on 37.2, 1988, pp. 225-228.

[24] L-T. Wang, E. J. McCluskey, “Linear feedback shift register design using cyclic codes”, Computers, IEEE Transactions on 37.10, 1988, pp. 1302-1306.

[25] R. A. Frohwerk, “Signature analysis: A new digital field service method”, Hewlett-Packard Journal 28.9, 1977, pp. 2-8.

[26] A. Y. Chan, “Easy-to-use signature analyzer accurately troubleshoots complex logic circuits”, Hewlett-Packard Journal, 1977, pp. 9-14.

[27] J. E. Smith, “Measures of the effectiveness of fault signature analysis”, Computers, IEEE Transactions on 100.6, 1980, pp. 510-514.

[28] O. Patashnik, “Circuit Segmentation for Pseudo-Exhaustive Testing”, Center for Reliable Computing, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, 1983.

[29] H. D. Schnurmann, E. Lindbloom, R. G. Carpenter, “The weighted random test-pattern generator”, Computers, IEEE Transactions on 100.7, 1975, pp. 695-700.

[30] A. J. Van de Goor, Z. Al-Ars, “Functional memory faults: a formal notation and a taxonomy”, in Proceedings IEEE VLSI Test Symposium, 2000, pp.281, 289.

[31] E. J. McCluskey, S. Bozorgui-Nesbat, “Design for autonomous test”, Computers, IEEE Transactions on 100.11, 1981, pp. 866-875.

[32] Z. Barzilai, D. Coppersmith, A. Rosenberg, “Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing”, Computers, IEEE Transactions on , vol.C-32, no.2, 1983, pp. 190,194.

[33] F. Elguibaly, M. W. El-Kharashi, “Multiple-input signature registers: an improved design”, Communications, Computers and Signal Processing, in Proceedings IEEE Pacific Rim Conference on. Vol. 2, 1997, pp. 519-522.

[34] M. Bushnell, V. Agrawal, “Essentials of electronic testing for digital, memory, and mixed-signal VLSI circuits”, Vol. 17. Springer, 2000, pp. 57-58, 465-485, 549-572.

[35] L.-T. Wang, C.-W. Wu, X. Wen, “VLSI test principles and architectures: design for testability”, Morgan Kaufmann Pub, 2006, pp. 271-289.

[36] K. K. Saluja, “Linear Feedback Shift Registers Theory and Applications”, Department of Electrical and Computer Engineering, University of Wisconsin-Madison, 1987, pp. 4-14.

[37] H. J. Wunderlich, G. Kiefer, “Bit-flipping BIST”, in Proceedings 1996 IEEE/ACM international conference on Computer-aided design, IEEE Computer Society, 1997, pp. 337-343.

[38] N. A. Touba, E. J. McCluskey, “Bit-fixing in pseudorandom sequences for scan BIST”, Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 20.4, 2001, pp. 545-555.

[39] L. T. Wang, E. J. McCluskey, “Complete feedback shift register design for built-in self-test”, Stanford University, Computer Systems Laboratory, 1986, pp. 56-59.

[40] K. D. Wagner, C. K. Chin, E. J. McCluskey, “Pseudorandom Testing”, Computers, IEEE Transactions on , vol.C-36, no.3, 1987, pp. 332,343.

[41] A. K. Pramanick, S. M. Reddy, “On the detection of delay faults”, in Proceedings International Test Conference, New Frontiers in Testing, 1988, pp. 845, 856.

[42] S. Wang, “Low hardware overhead scan based 3-weight weighted random BIST”, in Proceedings International Test Conference, 2001, pp. 868-877.

[43] J. G. Udell Jr, E. J. McCluskey, “Pseudo-exhaustive test and segmentation: Formal definitions and extended fault coverage results”, Fault-Tolerant Computing, FTCS-19,

Digest of Papers, in Proceedings IEEE International Symposium, 1989, pp. 292 – 298.

[44] J. G. Udell, E. J. McCluskey, “Partial hardware partitioning: a new pseudo-exhaustive test implementation”, in Proceedings International Test Conference, New Frontiers in Testing, 1988, pp. 1000.

[45] C. P. Gomes, H. Kautz, A. Sabharwal, B. Selman, (2008). Satisfiability solvers, Foundations of Artificial Intelligence, 3, 2008, pp. 89-134.

[46] L.-T. Wang, Y.-W. Chang, K. T. T. Cheng, “Electronic design automation: synthesis, verification, and test”, Morgan Kaufmann, 2009, pp. 903-905.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Anastasia Sannikova

17.05.2013