Test Point Insertion to improve BIST performance, and to reduce ATPG test time and data volume

Test Point Insertion to improve BIST performance, and to reduce ATPG test time and data volume

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.dr.ir. J.T. Fokkema,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen

op maandag 19 mei 2003 om 16:00 uur

door

Marc Jeroen GEUZEBROEK

elektrotechnisch ingenieur
geboren te Rotterdam

Dit proefschrift is goedgekeurd door de promotor:

Prof. dr. ir. A.J. van de Goor

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter	Technische Universiteit Delft
Prof. dr. ir. A.J. van de Goor, promotor	Technische Universiteit Delft
Prof. dr. ir. L.K. Nanver	Technische Universiteit Delft
Prof. ir. M.T.M. Segers	Technische Universiteit Eindhoven
Prof. dr. H. Corporaal	Technische Universiteit Eindhoven
Prof. dr. ir. Th.Krol	Universiteit Twente
Dr. ir. H.G. Kerkhoff	Universiteit Twente
Dr. ir. H.P.E. Vranken	Philips Research, Eindhoven
Prof. dr. C.I.M. Beenakker, reservelid	Technische Universiteit Delft

This work has been supported and funded by Philips Semiconductors

Published and distributed by: DUP Science

Delft University Press
P.O. Box 98
2600 MG Delft
The Netherlands
Telephone: +31 15 27 85 678
Telefax: + 31 15 27 85 706
E-mail: info@library.tudelft.nl

ISBN 90-407-2412-1

Keywords: Digital systems testing, ATPG, BIST, TPI

Printed in The Netherlands

Contents

Abstract

Test Point Insertion to improve BIST performance, and to reduce ATPG test time and data volume

The main subject of this dissertation is to facilitate structural testing by means of Test Point Insertion (TPI) for both on-chip and off-chip tests.

Efficient production testing is frequently hampered because current complex digital designs require too large test sets, even with powerful ATPG tools that generate compact test sets. An alternative is Built-In Self-Test (BIST); by embedding the test on-chip, expensive test equipment costs and test time can be reduced. However, BIST approaches often suffer from fault coverage problems, due to random pattern resistant faults. These problems can successfully be reduced, or even eliminated, by means of Test Point Insertion (TPI). In this dissertation we analyze three state-of-the-art TPI methods on their fault coverage improvement for BIST and develop a novel TPI algorithm that results in even better fault coverage improvement. This novel TPI algorithm has not only been developed to be applicable to Boolean circuits, but also to three-state designs. Results of several ISCAS and Philips industrial benchmark circuits show that the proposed TPI algorithm is applicable to both small Boolean circuits as well as to large complex industrial designs.

TPI for BIST not only improves PR fault coverage; it will be shown that TPI also results in more compact ATPG test set sizes. In this dissertation new TPI methods are presented that are aimed at solving ATPG specific testability problems, such that the ATPG test set sizes and CPU times can be reduced much further. These TPI methods are not only applicable to SAF ATPG; it has been demonstrated that also for gate-delay fault ATPG significant ATPG test set size reduction, ATPG fault coverage improvement and ATPG CPU time reduction can be achieved.

The new TPI methods have been implemented as part of the Delft Advanced Test Generation System and AMSAL, and are currently being used within Philips as part of their logic test tool set.

Preface and acknowledgments

This research on facilitating testing started in 1996 during my masters project at the Computer Engineering (CE) group of the department of Information Technology and Systems at the Delft University of Technology. Because integrated circuits can contain more and more transistors and testers become more and more expensive, we wanted to investigate whether the existing on-chip test methods were applicable to large industrial circuits and if we could integrate them in the test tools developed at Delft University, i.e., integrate them in the Delft Advanced Test generation system (DAT). At the same time, a Ph.D. position was made available by Philips Semiconductors at our test group on investigating ATPG driven BIST. One of the big issues with BIST was low fault coverage or high silicon overhead due to Random Pattern Resistant faults. As Test Point Insertion (TPI) is a solution for this problem, our first goal was the investigation and implement a TPI algorithm for improving BIST fault coverages.

During the research and the development of a TPI algorithm for BIST, Philips Semiconductors Hamburg got involved. They experienced more and more problems with growing ATPG test set sizes and they wanted to know if TPI could also be used to reduce these test set sizes. At that moment we shifted our focus on the investigation and development of TPI for facilitating ATPGs in generating compact test sets. Fortunately we ware able to reduce these ATPG test sets significantly by developing new TPI techniques aimed at facilitating ATPG. The results of the research and the development of TPI for BIST and ATPG are found in this dissertation.

Many people have contributed to this work and I would like to thank all of them. In particular, I will take the opportunity to thank the following persons for their contributions: First of all, I want to thank Hans van der Linden and Ad van de Goor for their support and guidance during both my masters and Ph.D. study and the preparation of this dissertation, and all the constructive technical discussions Thanks to Mario Konijnenburg for his support with DAT and the DAT source code. Also many thanks to the people from Philips who supported me with this research: Rene Segers who directed the project on Philips side; Friedrich Hapke for directing my research to TPI, in order to facilitate ATPG, and for providing the industrial circuits for the development, testing and optimization of a TPI algorithm for the “real-world”; and Harald Vranken for his support, interest and the constructive technical discussions on TPI.

I would also like to thank all people of the Computer Engineering group at the Delft University of Technology, especially thanks to Hans van der Linden, Mario Konijnenburg and Steven Roos for being fine colleagues with whom I shared the office. Also many thanks to Bert Meys for providing and maintaining the computer network environment.

Finally, of course, I would like to thank all my friends and my family for their support during all years of this Ph.D. study.

CHAPTER 1

Introduction

During the fabrication of *Integrated Circuits (ICs)*, even the smallest irregularity, like a dust-particle, can result in a malfunctioning IC. The testing of ICs forms an important step in the process of preventing that these malfunctioning ICs will be delivered to consumers or are used in end-products. It also is an important feedback for optimizing the manufacturing process. Customers of the IC-manufacturers want a guarantee that the ICs that are shipped to them are of high quality. They only allow a low number of malfunctioning ICs. The *Defect Level (DL)* [Wil81] is a measure for the fraction of malfunctioning ICs that pass all tests and can be calculated with Eq. 1.1,

$$DL = 1 - Y^{1-FC} \tag{1.1}$$

in which Y is the *process yield*, defined as the fraction of manufactured parts that is defect free, and FC is the *Fault Coverage*, defined as the ratio of the number of actual detected faults and the total number of faults in the IC, whereby the faults are assumed of a particular *Fault Model* as described in Section 1.1. The DL is often expressed in the *DPM (Defects Per Million)* (defective parts per million shipped instances). In order to reach low DPM-levels, the manufacturers need high quality tests. E.g., assume for a specific design a yield of 0.9 (90%) and a DL of 200 DPM (DL=0.0002), the required fault coverage can be calculated with

$$FC = 1 - \frac{log(1-DL)}{log\, Y} \tag{1.2}$$

and becomes for this example 99.81%. Hence at least 99.81% of all faults should be detected otherwise more than 200 out of 1 million parts contain defects.

During the complete design and manufacturing process, different levels of testing are required:

1. Functional testing:
   Verification whether the design/IC meets its functional specification. Does it do what it is supposed to do?

2. Structural testing:
   Verification whether the design/IC meets its structural specification. Is the layout as specified and are there no defects introduced?

3. Application mode testing:
   Verification whether the design/IC meets the specifications of the environment in which it is used.

Testing in this dissertation refers to structural testing; i.e., testing whether the IC has been manufactured correctly. For example, whether no *spot-defects* are introduced. Spot-defects are local conducting (i.e., shorts) or non-conducting (i.e., opens) disturbances in the IC silicon structures. These disturbances can lead to an adjusted behavior of an IC, i.e., to incorrect logic functions, to timing problems, to higher power consumption, etc. The structural test has to detect this adjusted behavior.

An IC usually consists of logic, memory and possibly mixed-signal blocks. Memories have a very regular structure and can be tested for defects by regular algorithms in polynomial time; e.g., by *march tests* [vdG98]. The logic blocks usually do not have a regular structure and more sophisticated algorithms are necessary to generate tests. Besides digital logic, mixed-signal blocks also contain analog logic. Mixed-signal circuits are out-of-scope of this dissertation.

Logic circuits are tested by the application of a set of *test patterns*. A test pattern consists of a single set of simultaneous applied input values, i.e., stimulus, that is applied to the circuit, and a set of expected output values, i.e., response, for a defect-free circuit. Often, when speaking of test patterns, only the stimuli are meant. A defect in a circuit is detected by a test pattern when after applying the stimulus to the circuit, the response for that IC differs from the expected response. The complete application of the set of test patterns; i.e., the *test set*, to check for defects in the IC is called a *test* for this IC.

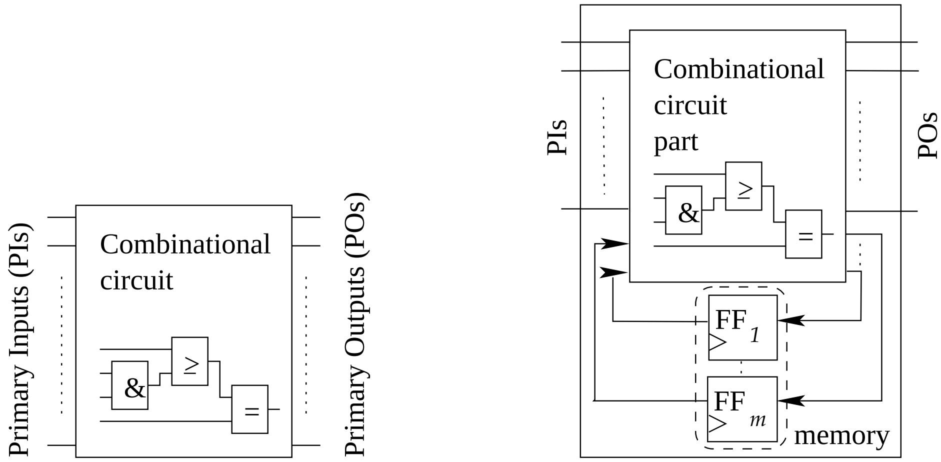
Logic circuits can be divided into *combinational circuits*, see Fig. 1.1(a) and *sequential circuits*, see Fig. 1.1(b):

Combinational circuits: A combinational circuit, see Fig. 1.1(a), consists of an interconnected set of gates, containing no feedback loops[1][Kon98] with a set of *Primary Inputs (PIs)* and a set of *Primary Outputs (POs)* to interact with the circuit. The output response of a combinational circuit only depends on the *current* applied stimulus.

Sequential circuits: A sequential circuit, see Fig. 1.1(b), consists of a combinational circuit part and feedback loops containing memory elements, which give the circuit the capability to memorize information [Kon98]. The output response of a sequential circuit depends on the current applied stimulus *and* on the stimuli applied in the

---

[1] A feedback loop is a directed path from the output of a gate to an input of that gate.



(a) Combinational

(b) Sequential

Figure 1.1: Combinational and sequential logic circuits

past. This information from the past is usually stored in explicit memory; i.e., in one or more *latches* or *flipflops (FFs)*.

Most ICs contain sequential circuits, implemented with FFs and/or latches. Testing combinational circuits is far easier than testing sequential circuits, because the stimuli applied in the past do not have to be taken into account. A sequential circuit can be tested as a combinational circuit when the circuit conforms to a *full-scan* design [Eic77]. In a full-scan design, shown in Fig. 1.2, all FFs are made *scannable* by linking them together to form one or more shift-registers, the *scan-chain(s)*. Bits are shifted in the scan-chain through the *scan-in* input, and are shifted out of the scan-chain through the *scan-out* output. This way all FF values can be controlled (observed) directly by shifting in (out) values to (from) the scan-chain. The FFs that form the scan-chain are called *scan flipflops (SFFs)*. The *test-mode* signal specifies whether the FFs work in normal application mode or in scan mode. In this dissertation it is assumed that the designs conform to full-scan designs.

1.1 Fault models

As previously mentioned, defects can result in adjusted behavior in different ways. To model this behavior, various fault models are introduced. In this dissertation we will focus only on the following two important fault models: the *stuck-at fault (SAF)* model and the *delay fault* model.

Stuck-at fault: The defect results in signal lines in the circuit being stuck-at a fixed

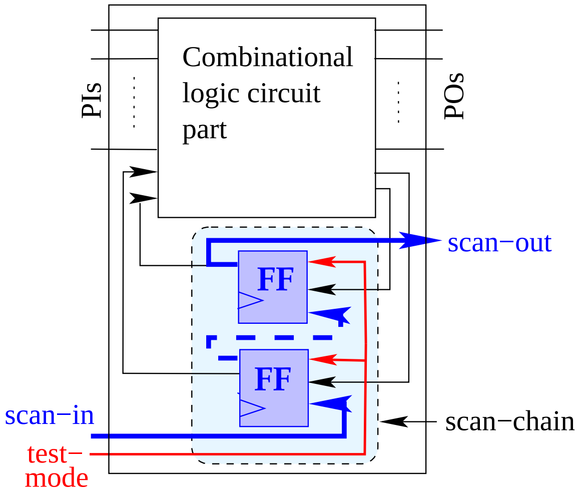


Figure 1.2: A sequential design with a scan-chain

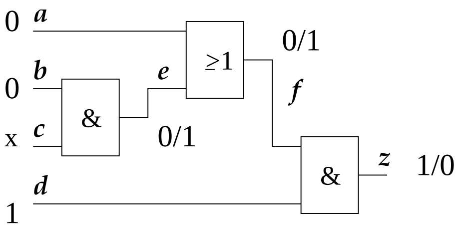


Figure 1.3: Circuit with a SA1 fault at line *e*

value, i.e., *stuck-at logic 0 (SA0)* or *stuck-at logic 1 (SA1)*. A test pattern for detecting a SAF at a signal line *l* has to *activate* the fault, i.e., the applied stimulus should result in a different value on *l* in the fault-free case compared to the faulty case, and it should *propagate* the fault-effect to an output. Fig. 1.3 illustrates this for a simple example circuit with four PIs (a to d) and one PO (z) with a SA1 fault at internal line e. The fault is activated by the 0 on PI b; i.e., e is 0 in the fault-free circuit and 1 the faulty circuit, shown as 0/1 in Fig. 1.3. The 0/1 fault-effect at e is propagated to PO z by the 0 at a and the 1 at d. PO z is 0 in the good case and 1 in the faulty case (0/1 in Fig. 1.3). The value of PI c does not matter, shown with the don't-care value 'x'.

Delay fault: The defect results in malfunctioning of the IC at a given clock-frequency within the timing specification of the IC. Delay faults can be divided into two groups [Smi85]:

1. **Path-delay fault:** A path is defined faulty if it fails to propagate a transition from the path input to the path output, within a specified time interval.

2. **Gate-delay fault:** A gate is defined faulty if its gate defect results in at least one path-delay fault.

Besides the stuck-at fault and the delay fault, other fault models exist, such as the I_{DDq} fault, the stuck-open fault and the bridge-fault [Fuj85].

1.2 Testing methods

Testing of a circuit can take place in two different ways:

1. *Off*-chip testing
2. *On*-chip testing

Subsection 1.2.1 describes off-chip testing while Subsection 1.2.2 describes on-chip testing. The main advantages and disadvantages are summarized in Subsection 1.2.3.

1.2.1 Off-chip testing

In case the circuit is tested off-chip, external *Automated Test Equipment (ATE)* (i.e., a tester) is used to apply the test stimuli to the PIs and to compare the output responses of the *Circuit-Under-Test (CUT)* with the responses of the fault-free circuit. Before the test patterns can be applied to a circuit, they have to be generated and stored in the tester. These test patterns are generated by *Automatic Test Pattern Generators (ATPGs)*.

Given a circuit with n inputs (both PI and SFF inputs), 2^n different test patterns are possible. One can imagine that it is not feasible to apply all possible test patterns to large circuits (with a large number of inputs). Therefore the ATPG has to generate a subset of test patterns with which still all possible SAFs can be detected. Nowadays state-of-the-art ATPGs [vdL96, Wai90] are capable of generating test sets with a nearly complete coverage of all detectable SAFs, even for the larger and more complex circuits. But with the increasing complexity of circuits, the ATPG test set sizes also grow, even with all state-of-the art techniques [Ake87, Goe81, Tro91, Pom91, Cha92, Kon96c] that aim at producing a compact test set. This may have a major impact on the test costs for the semiconductor industry. Increasing test set sizes not only result in longer test times, but also in increased memory usage on the ATE in order to store the test patterns and the output responses.

Due to the technological progress, the clock-frequencies on which ICs run are ever increasing. To be able to detect possible timing-problems, it can be important that the IC is tested *at-speed*, i.e., it is tested at the clock frequency it will run in normal application mode. As a result, for new ICs often new testers are required which are able to operate at these high frequencies and which are accurate enough to capture the responses of the ICs.

To cope with this increasing complexity of circuits, faster testers with much more memory will be required, which will become very expensive.

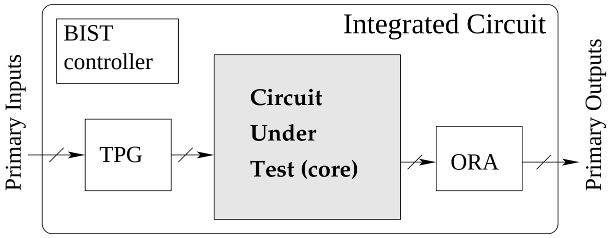


Figure 1.4: On-chip testing

1.2.2 On-chip testing

With on-chip testing is meant that the test is embedded on the IC. A circuit with an on-chip test is often referred to as a circuit with *Built-In Self-Test (BIST)*. The basic idea of a circuit with BIST is depicted in Fig. 1.4. Both the *Test Pattern Generator (TPG)*, which applies the test stimuli to the inputs of the core, and the *Output Response Analyzer (ORA)*, which compares the output responses with the responses of a fault-free circuit, are embedded on the IC. They are controlled by a BIST controller. The advantage of a circuit with BIST is that there is no need for expensive external testers to apply the input stimuli to the circuit and to analyze the output responses. This opens the possibility for massive parallel testing.

One of the main draw-backs of BIST is the silicon-overhead necessary to implement the BIST hardware, i.e., the TPG, the ORA and the BIST controller. This overhead depends on the type of TPG that is used. Often a *Linear Feedback Shift Register (LFSR)*, described in Chapter 2, is used as *Pseudo Random (PR)* TPG to generate the test patterns. This is a very low cost implementation, but the fault coverages that can be achieved with this TPG are often also low. In order to get a higher fault coverage, the PRTPG should apply a very large number of test patterns, resulting in impractical test application times.

Other BIST solutions embed deterministic (ATPG) test patterns in the TPG. An easy implementation would be embedding a ROM in which a complete ATPG generated test set is stored. Although a high fault coverage can be achieved, the silicon overhead for the ROM can be significant.

Another draw-back of BIST is the possible performance-penalty introduced by the extra BIST hardware, adding the extra hardware to a design can result in extra development iterations and increases the development time.

More detailed descriptions of BIST methods are given in Chapter 2.

1.2.3 Advantages and disadvantages of off-chip and on-chip testing

The advantages (+) of off-chip testing are:

+ High fault coverages can be achieved.

+ Relative compact test sets, resulting in relative low test application times.

+ No performance impact on circuit for normal application mode, except for scan-chain insertion.

while the disadvantages (-) of off-chip testing are:

- Due to ATE limitations, often at-speed testing is not possible.
- Expensive external testers are necessary.
- ATPG can take much CPU-time to generate the test set.

The advantages (+) of on-chip testing are:

+ No expensive external testers are necessary.
+ Often no complex ATPG tool is necessary to generate the test set.
+ At-speed test with a possibility for parallel testing.

and the disadvantages (-) of on-chip testing are:

- Lower fault coverages, and/or
- High silicon overhead.
- Hardware insertion may result in extra development iterations in the design flow; i.e., longer development times.
- A possible performance-penalty.
- Fault localization (diagnostics) is difficult.

The ever increasing complexity of designs has a major impact on the test costs. Not only faster testers are required which still can do accurate measurements, but the increasing ATPG test set sizes result in higher test pattern memory requirements and longer test application times. The more (tester) time a test takes for an IC, the more expensive the test becomes. According to the International Technology Roadmap for Semiconductors, it is even expected that without the necessary solutions, the ATE will not be able to cope with these high demands within only a few years [Sem01].

BIST would be a solution to decrease the tester demands, but the lower fault coverage achieved by BIST often does not meet the high quality demands of the semiconductor industry, or the test time required to reach a high fault coverage is way too long. Also the silicon area overhead penalty and/or the performance penalty and the diagnostics problem are known reasons why BIST often is not used as alternative to ATPG/external ATE. When a test fails with on-chip testing it is also difficult to find the fault location, as it is in most cases not known which pattern(s) caused the test failure, and hence which faults are covered by the failing pattern(s). In other words, diagnostics is difficult with on-chip testing. In this dissertation methods are proposed which can be used to reduce test set sizes for off-chip testing, and methods which can be used to improve the fault coverages for on-chip testing without large silicon overhead.

1.3 Test Point Insertion

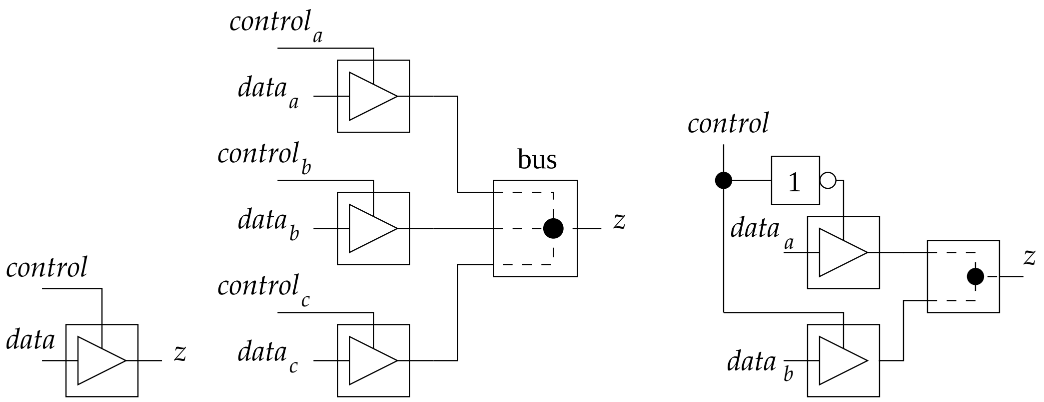
One way to solve test problems is by inserting *Test Points (TPs)* in the circuit. TPs provide extra inputs and/or outputs to internal parts of the circuit. TPs are often divided into *Control Points (CPs)* and *Observation Points (OPs)*. A CP provides an extra input to the circuit while an OP provides an extra output. With a CP, internal signal lines can be set to a specific value such that it becomes easier to *activate* faults in the fan-out cone of the CP. An OP provides an extra output. At an OP, internal parts of the circuit can directly be observed. This way fault-effects from faults in the fan-in cone of the inserted OP do not have to be propagated further through the CUT, but can be observed directly at the extra inserted output.

With the extra inputs and outputs of inserted TPs, it becomes easier to detect faults within the circuit that were hard-to-test before. *Test Point Insertion (TPI)* can result in higher fault coverages achievable with on-chip testing and can also reduce ATPG test generation times and test set sizes for off-chip testing.

However, not every position in the circuit is suitable for a TP. The problem for TPI is to find the positions in the circuit where TPs will result in the best fault coverage improvement, or the best test set size or test generation time reduction. There exist several TPI methods which search the positions in CUTs at which TPs would result in good fault coverage improvement for on-chip testing. However, most of these methods are designed for Boolean circuits, while the ’real-world’ industrial circuits also contain non-Boolean elements and suffer from other restrictions that these TPI methods cannot cope with. TPI methods for off-chip testing, in order to reduce ATPG test times and data volume, have not been researched. In this dissertation TPI methods are proposed which can be used to increase both the fault coverages achievable with on-chip testing and to reduce ATPG test times and data volume with off-chip testing for *industrial* circuits.

1.4 Industrial circuits

Besides the well-known Boolean elements, e.g., AND-gates, OR-gates, inverters, etc., in the industry also non-Boolean elements are used. Besides 0 or 1, these elements can also result in the ’high impedance’ state or *Z*. Industrial circuits also often contain blocks, like embedded memories, from which it is not always known what the values on its outputs are. This results in ’unknown’ *U* (fixed) values on several inputs of the circuit core, the inputs that are connected with the outputs of that block (memory). In the remaining part of this dissertation, with Boolean circuits are meant circuits containing only Boolean gates and no unknown (fixed) circuit inputs. Subsection 1.4.1 describes the occurrence of Z and U values in three-state circuits, while Subsection 1.4.2 shortly describes several three-state elements found in industrial circuits. These three-state elements are taken from [vdL96].



(a) A bus-driver

(b) A three-state bus driven by three bus-drivers

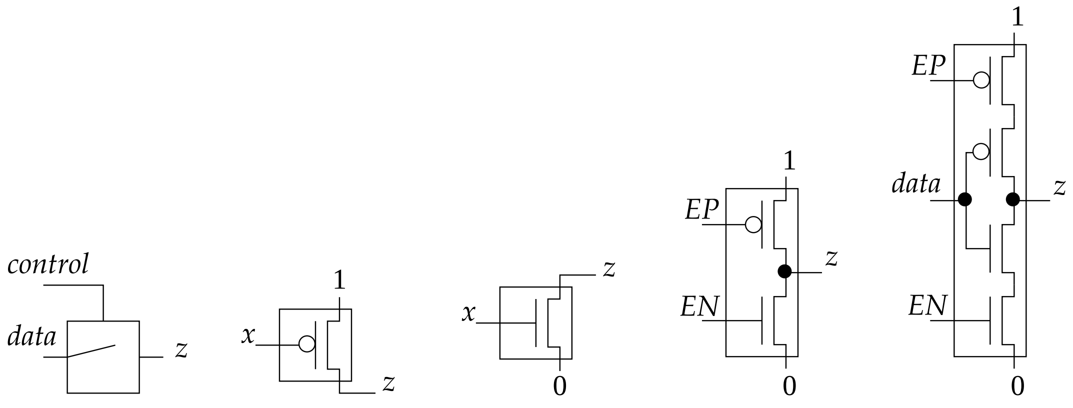
(c) The two bus-drivers are never enabled at the same time

Figure 1.5: Examples of three-state logic

1.4.1 The ’floating’ (Z) and ’unknown’ (U) value in three-state circuits

Often, circuits are designed with Boolean elements only and can be expanded into a circuit consisting of primitive Boolean elements. These primitive Boolean elements are listed in Appendix C. They are the **AND / NAND**, **OR / NOR**, **XOR / NXOR** and the **BUF (buffer) / INV (inverter)**. The output of a Boolean element is 0 or 1, depending on the input-values.

Besides Boolean logic, industrial designs can also contain non-Boolean logic. The output of these elements cannot only be 0 or 1, but can also be at high impedance (floating), commonly denoted by signal value ***Z***. These logic elements are called three-state elements. An example of such an element is the **bus-driver**, see Fig. 1.5(a). When a bus-driver is enabled (1 on the control-input), the output of the driver equals the data-input, when it is disabled (0 on the control-input), the output floats (is Z). Bus-drivers are used to drive buses; i.e., multiple bus-drivers are used to drive a single node, as depicted in Fig. 1.5(b). When two or more bus-drivers are enabled and drive the bus with opposite values, it is not known what the value at the bus-node will be. A bus-conflict occurs and the value at the bus node will be *unknown **(U)***. Besides that the logic value at the bus cannot be determined, bus-conflicts can also result in IC damage. Connecting two lines with opposite values can result in a short between power-lines. Therefore bus-conflicts should be avoided. Often this is accomplished by making sure that the bus-drivers connected to a given bus are never enabled at the same time. Fig. 1.5(c) shows an example how bus-conflicts can be avoided; the two bus-drivers are never enabled at the same time; assuming that the inverter operates properly.



(a) Switch (b) Open Drain PFET (c) Open Drain NFET (d) Tri (e) Tristate inverter

Figure 1.6: Three-state elements

1.4.2 Three-state elements

This subsection gives a short description of several three-state elements found and used in the semiconductor industry. More information on these three-state elements can be found in Appendix D.

(N)Switch: This element, shown in Fig. 1.6(a) has two inputs. A *control* and a *data* input. When the control input is 1 (0 in case of an NSwitch), the output equals the value of *data*, else the output floats and is Z.

(N)Bus-driver: Is the same as (N)Switch except that it cannot propagate Z-values from the data input to the output. In that case the output becomes unknown.

Three-state bus: The three-state bus, shown in Fig. 1.5(b) is driven by multiple lines (inputs). When all lines are undriven, i.e., propagate value Z, the output will also be Z. When only one line is driven, the output equals the value of this line. But when multiple lines are driven, with one line carrying the value 0 and another the value 1, a bus-conflict occurs and the output becomes unknown.

Wired AND (WAND): This kind of bus can be seen as an AND gate. When all inputs are undriven, the output will be Z. If none of the inputs carry the value 0 and at least one input carries a 1, the output becomes 1. A 0 on an input dominates all other input values and the output becomes 0.

Wired OR (WOR): See Wired AND, except that this bus can be seen as an OR gate; an input value 1 dominates input value 0.

Pull-down bus: Identical to the three-state bus except for the case that no lines are driven, i.e., all are Z. The output does not become Z but is pulled-down to a 0.

Pull-up bus: Identical to the three-state bus except for the case that no lines are driven, i.e., all are Z. The output is pulled-up to a 1 instead of Z.

Open Drain PFET (ODP): The ODP, shown in Fig. 1.6(b) can be seen as a bus-driver which output is 1 when the input of this element is 0 and floats when the input is 1.

Open Drain NFET (ODN): The ODN, shown in Fig. 1.6(c) can be seen as a bus-driver which output is 0 when the input of this element is 1 and floats when the input is 0.

Tri: The Tri, shown in Fig. 1.6(d) is a combination of an ODP and an ODN in which the outputs are combined. The designer should prevent that both drivers are turned on at the same time.

Tristate inverter (TRINV): The Tristate inverter, shown in Fig. 1.6(e) is an element which consists of a bus that is driven by two drivers, an NBus-driver and a Bus-driver. The **data** input is used to enable only one of these drivers. This way no conflict can occur.

1.5 Open questions and problems

The increasing complexity of ICs makes testing of ICs more and more costly. In order to reduce the ATE costs, on-chip testing, i.e., BIST, is a way to embed the test in the IC chip reducing the need of expensive ATE. However the fault coverages achieved with BIST are often not high enough to reach an acceptable low DPM level for the semiconductor industry. The open questions that still exist are:

1. What kind of methods exist in literature that can be used to improve fault coverage with on-chip testing?
2. How well are existing on-chip test methods in achieving both low hardware overhead and high fault coverage?
3. Are existing on-chip test methods applicable to *industrial* circuits, i.e., can they cope with high-impedance and unknown values and are they applicable to very large designs?

By inserting TPs into a circuit, it becomes easier to detect hard-to-test faults. After TPI, higher fault coverages can be achieved with BIST. The questions that arise are:

1. Which TPI methods can be used to improve the fault coverage with on-chip testing?
2. Are existing TPI methods applicable to *industrial* circuits with respect to

- the fault coverage achievable after TPI?
- the hardware overhead of TPs? (how many TPs are inserted in order to get higher fault coverage?)
- the circuit size?
- the element types in CUT, i.e., Boolean and non-Boolean element?

Applying on-chip test, i.e., BIST, is only one way to reduce the ATE costs with respect to test pattern memory requirements and test application & generation times. TPI also makes it easier for ATPGs to generate test patterns for the faults in the circuit. But does this also result in significant reduction of test set sizes and so in test pattern memory requirements and test application generation times? The open question for TPI on facilitating ATPG are:

1. Can TPI (for BIST) be used to significantly reduce ATPG test set sizes?
2. Which ATPG specific test problems exist that cause large test sets?
3. Which TPI techniques can be used that aim at solving the ATPG specific test problems that cause large test sets?
4. Does TPI for ATPG, e.g., with TPI techniques that aim at solving the ATPG specific test problems, result in better test set size reduction compared to TPI for BIST?

Currently it is assumed that TPI is applied to improve the detectability of SAFs in the circuit. But what is the impact of TPI on other fault models? I.e., can TPI also be used to reduce ATPG test set sizes for delay faults?

1.6 Overview of this dissertation

This dissertation is organized as follows: At first, the goal of this research was the development of ATPG driven BIST. Therefore existing BIST methods have been studied on their performance with respect to fault coverage, silicon overhead and application to large industrial designs. During studying these existing methods, we found out that all of them suffered from low fault coverages and/or high silicon overhead for circuits with faults that are hard-to-detect by PR patterns. As TPI is a successful technique to improve the testability of circuits, the research was continued on TPI instead of BIST. For completeness, **Chapter 2** provides the research and evaluation of the existing state-of-the-art BIST methods with respect to the hardware overhead and achieved fault coverage. **Chapter 3** starts with a general description of the concept, purpose and properties of TPI, followed by descriptions of state-of-the-art TPI algorithms found in literature, which are mainly aimed at improving the PR fault coverage in a BIST environment. This chapter also provides an overview of the TPI topics addressed in the remaining part of this dissertation including an overview of the test circuits used to benchmark TPI algorithms.

The state-of-the-art TPI methods often are not suitable for industrial circuits, due to CPU time consumption, accuracy of the selection of TP positions, and/or support for industrial circuits with three-state elements and U values. **Chapter 4** provides a proposal for a TPI algorithm that results in higher fault coverage improvements for BIST than the existing TPI methods and can be applied to industrial circuits. **Chapter 5** starts with the impact of TPI for PR BIST on ATPG test set sizes, followed by a description of the ATPG specific testability problems that cause large tests. Given these ATPG specific testability problems, a proposal of a TPI algorithm for ATPG is given, i.e., a TPI algorithm that aims at reducing ATPG test set sizes. Experimental results will show the effectiveness of the proposed TPI algorithm for ATPG. **Chapter 6** shows the impact of TPI on ATPG test set sizes for delay faults. The chapter starts with describing the similarities & differences between SAF ATPG and delay fault ATPG, including the implications of delay fault ATPG for TPI. It will be shown that TPI for ATPG can be used both for SAFs and for delay faults. **Chapter 7** provides a summary and conclusions of the main contributions of this dissertation, including suggestions for future work.

Appendices A and **B** provide details on the ISCAS, respectively the industrial (Philips), circuits used throughout this dissertation for experimental results. **Appendices C** and **D** provide details on the Boolean, respectively non-Boolean (three-state), elements that are found in (industrial) circuits. **Appendix E** provides information on the test tools DAT and AMSAL[2], that have been used. In these tools the algorithms and techniques presented in this dissertation have been implemented.

---

[2] DAT is embedded in AMSAL

CHAPTER 2

Built-In Self-Test

This chapter gives an overview of Built-In Self-Test (BIST) techniques that can be used to implement on-chip testing. Although in Section 1.2.2 already a short description of BIST has been given, Section 2.1 starts with describing the general concept and purpose of BIST; what is BIST and what can you do with it? Section 2.2 describes the Linear Feedback Shift Register (LFSR) and the Multiple Input Shift Register (MISR), which are often found in BIST implementation as TPG, respectively ORA. There are restrictions in order to be able to implement an industrial circuit with BIST. The restrictions, limitations and implications of BIST on industrial circuits are summarized in Section 2.3. BIST is often characterized by its TPG implementation. Section 2.4 starts with describing the different types of TPGs that are used to implement BIST. It is followed with techniques that can be used to facilitate the TPG to generate patterns that do detect the hard-to-test faults. Sections 2.5 to 2.7 describe state-of-the-art BIST methods, based on the different BIST facilitation techniques listed in Section 2.4; Section 2.5 describes BIST methods that modify the test patterns generated by the TPG, Section 2.6 describes BIST methods that fully embed a deterministic test set, and Section 2.7 describes BIST methods that only embed parts of deterministic test sets or test patterns. Sections 2.8 describes how the circuit itself itself can be made better testable, instead of using state-of-the-art BIST methods. Section 2.9 concludes this chapter.

2.1 Concept and purpose of BIST

The purpose of BIST is to embed the test for a circuit on-chip. This way the external tester requirements, and hence the test costs for the semiconductor industry, can be reduced. BIST is accomplished by adding logic to the IC that allows the on-chip testing of the IC, hence no stimuli have to be applied by the ATE, neither the CUT responses have to be captured by the ATE and compared with the responses of a fault-free design. The general BIST operation is depicted in Fig. 2.1. This is a slightly more detailed version of the picture shown in Fig. 1.4.

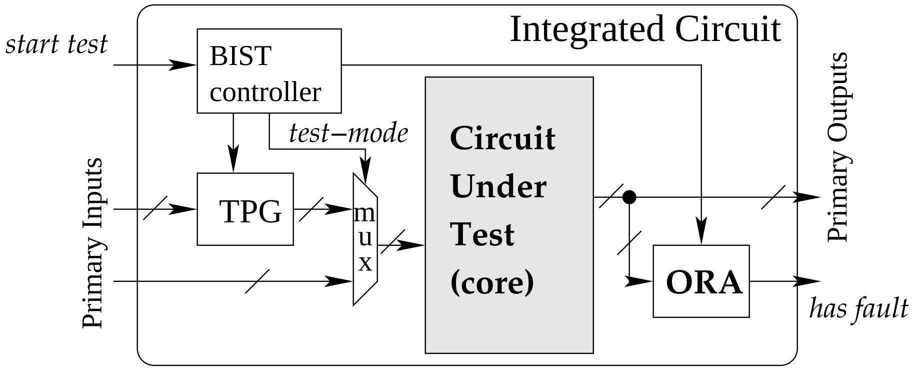


Figure 2.1: Built-In Self-Test

In normal application mode, the input values of the CUT are applied externally through the PIs. In test mode, entered by setting *start test*, the BIST controller sets the *test-mode* signal and instead of external input values, the TPG applies the stimuli to the CUT. It is possible that the TPG is initialized by an initial pattern applied externally (through PIs). In test mode, the ORA captures the output values from the CUT. After all test patterns are applied to the circuit and all responses have been captured by the ORA, the signal *has fault* indicates whether a fault has been detected.

2.2 TPG and ORA

LFSRs are often used in BIST as TPG, while *Multiple Input Shift Registers (MISRs)* are often used as ORA in BIST. Therefore, before the descriptions of existing BIST implementations in Sections 2.5-2.7, first the LFSR and MISR are described in Subsections 2.2.1 and 2.2.2.

2.2.1 Linear Feedback Shift Register (LFSR)

BIST TPGs are often implemented by means of an LFSR[Bar87]. The general structure of an LFSR is depicted in Fig. 2.2. An LFSR is composed of R memory cells (cells M_0 to M_{R-1} in Fig. 2.2) connected together as a shift register with linear feedbacks (exclusive or (*exor*) function). The variables h_r, with $0 \leq r \leq R-1$, the feedback coefficients, indicate whether there exists a feedback connection from the output of memory cell M_{R-1} to the input of memory cell M_r. These coefficients can be 0 (no feedback connection) or 1 (feedback connection).

The next state of an LFSR is uniquely determined from the previous state by the feedback network. Given an initial state, or *seed*, the LFSR will generate a sequence of different states, as illustrated by the LFSR given in Fig. 2.3. As soon as a state repeats,

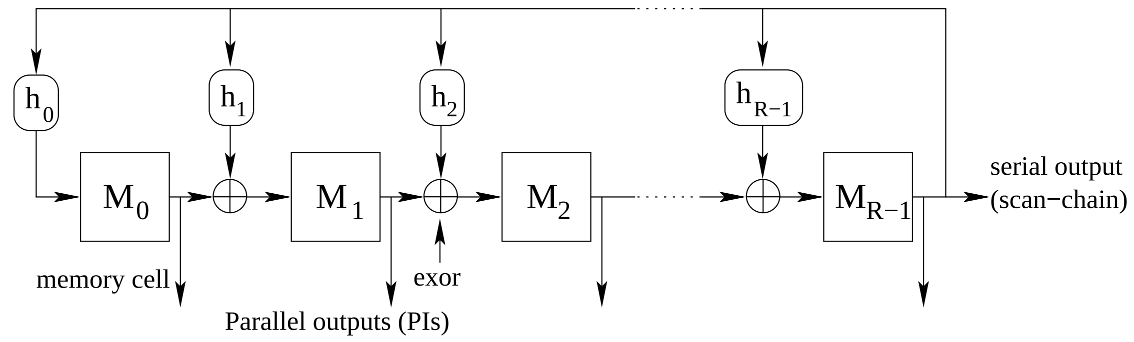
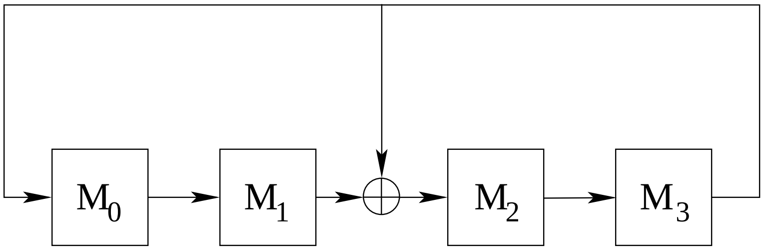


Figure 2.2: Linear Feedback Shift Register



M_0	M_1	M_2	M_3
0	1	1	0
0	0	1	1
1	0	1	1
1	1	1	1
1	1	0	1
1	1	0	0
0	1	1	0

Figure 2.3: A 4-bit LFSR

all following states will also repeat; the sequence of states is periodic. Given the seed of `0110`, the 4-bit LFSR in Fig. 2.3 will generate a sequence of six different states. When the parallel outputs of the LFSR are connected to the PIs of the CUT, this LFSR would apply 6 of the 16(=2^4) possible test patterns to the CUT. LFSRs can also apply PR patterns to scan-chains. In that case the last memory cell is often used as serial input for the scan-chain.

An LFSR can be characterized by its *characteristic polynomial (ChPol)*. The ChPol of an R-bit LFSR is given in Eq. 2.1.

$$ChPol(x) = h_0 + h_1 \cdot x + h_2 \cdot x^2 + \cdots + h_{R-1} \cdot x^{R-1} + x^R \tag{2.1}$$

A ChPol is called a *primitive* ChPol when the corresponding R-bit LFSR would generate a sequence of $2^R - 1$ different states, given any non-zero seed. This is also called a *maximum length sequence* and the corresponding LFSR is called a *maximum configured LFSR*. Because an LFSR is *linear*, an LFSR initialized with the all-zero pattern will always generate the all-zero pattern.

The next state of the LFSR can be calculated by multiplying the current state of the

$$\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \\ h_0 & h_1 & h_2 & \dots & h_{R-1} \end{bmatrix}$$

Figure 2.4: The characteristic matrix of an LFSR

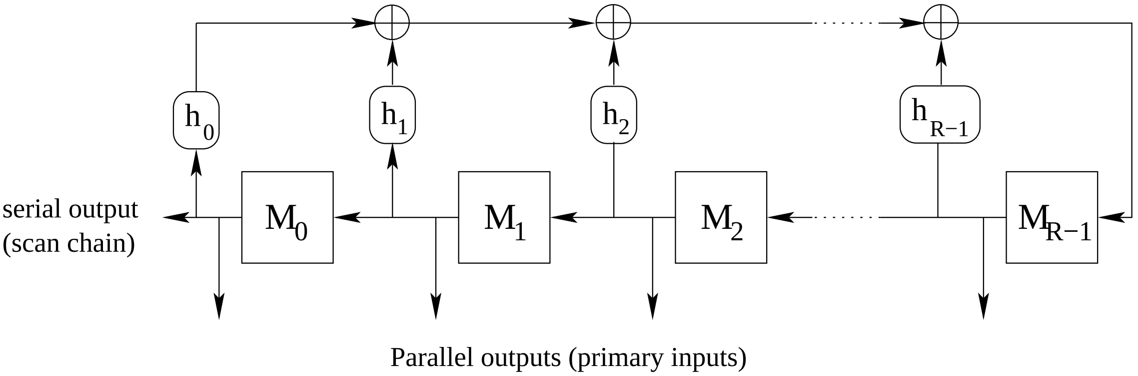


Figure 2.5: Out-tapping (type-I) Linear Feedback Shift Register

LFSR with the characteristic matrix, T_c, shown in Fig. 2.4. Given the current state $X(t)$, the next state, $X(t+1)$ can be calculated with Eq. 2.2.

$$X(t+1) = X(t) \cdot T_c \tag{2.2}$$

or given the initial state $X(0)$, the state after t cycles is given by:

$$X(t) = X(0) \cdot T_c^t \tag{2.3}$$

The ChPol given in Eq. 2.1 can be used to describe both an *out-tapping* (or *type-I*) LFSR and an *in-tapping* (or *type-II*) LFSR. The LFSR depicted in Fig. 2.2 is an in-tapping LFSR; the XOR taps are within the shift register. The out-tapping LFSR represented by the same ChPol is depicted in Fig. 2.5. The in-tapping LFSR is more commonly used than the out-tapping LFSR, but both configurations are found in existing BIST schemes.

2.2.2 Multiple Input Shift Register (MISR)

As ORA often the Multiple Input Shift Register (MISR) [Bar87] is used. The MISR is depicted in Fig. 2.6. A MISR is quite similar to an in-tapping LFSR, it consists of R memory cells ($M_0 \dots M_{R-1}$) with linear feedbacks from cell M_{R-1}. However, the MISR is also fed by the POs of the CUT. Like the LFSR, the MISR can be characterized by the

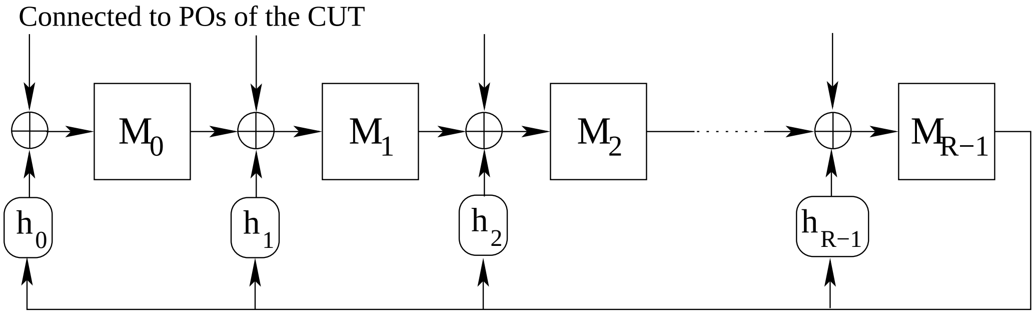


Figure 2.6: A Multiple Input Shift Register

polynomial given in Eq. 2.1. Given the ChPol and the initial state of the MISR, every cycle the MISR will generate a new state, based on the current state, the ChPol and the response of the CUT. The state of the MISR is referred to as the *signature*. Given the initial state, the ChPol and the set of responses from the CUT to a given test, the signature of the MISR after the complete test can be determined. This signature is compared to the signature of a simulated fault-free circuit. When these signatures do not match, the CUT is not fault-free. However, it is possible that due to *aliasing*, the signature of the CUT matches the signature of the fault-free circuit, while not all responses were correct. In [Bar87] it is shown that the probability on aliasing can be approximated with Eq. 2.4, where R is the number of stages in the MISR.

$$p(aliasing) \approx \frac{1}{2^R} \tag{2.4}$$

Eq. 2.4 shows that the probability on aliasing is independent of the number of test patterns. As long as the number of MISR cells is large enough; e.g., for $R_{MISR} \geq 16$, the probability of aliasing is almost negligible ($\leq 0.0015\%$).

2.3 BIST and real-world circuits

BIST is not straightforwardly suitable for implementation in every circuit. A circuit must meet several requirements, especially with respect to high impedance and unknown values, before BIST can successfully be applied.

As described in Section 2.2, MISRs are often used as ORA. In order to get a correct and known signature in the MISR, no 'unknown' or floating values are allowed to be shifted in the MISR during test. When such a value is shifted in the MISR, the complete MISR state becomes unknown and it cannot be checked whether the signature conforms to a fault-free circuit. Therefore a circuit should satisfy the following requirements for successful BIST implementation:

1. There are no floating/unknown circuit inputs that can propagate to the MISR.

2. POs that can float or be unknown should not be connected directly to the MISR in order to avoid an unknown MISR signature.

These requirements can be satisfied when:

- Embedded memory inputs, bi-directional inputs and other possibly floating inputs or inputs with an unknown value are set to a known value by extra test-logic.
- Bus-drivers of three-state buses are never enabled at the same time, regardless of the circuit's input values,in order to avoid bus-conflicts and hence buses becoming unknown.
- POs that can float, e.g., due to floating buses, should be pulled-up or pulled-down before their value is shifted into the MISR.
- POs that cannot be pulled-up/pulled-down and still can float or can become unknown should not be connected to the MISR. This has a negative impact on the achievable fault coverage, because an unknown MISR signature means no fault coverage at all.

Bus-conflicts should be avoided regardless whether they can result in unknown MISR signatures because they can cause circuit damage. ATPGs do avoid bus-conflicts by not generating test patterns that cause bus-conflicts, but PRTPGs do not! Because PRTPGs are often used in BIST implementations, one should make sure that buses in the circuit can never become conflicting regardless of the circuit input values.

In Gu et al. [Gu 01] techniques are described, to overcome BIST problems e.g., timing violations and unknown MISR signatures due to unknowns, that can be used to minimize the designs efforts with respect to BIST implementation in industrial designs.

In general, BIST implementations assume that the circuit confirms to a full-scan design. In order to be able to reach high enough fault coverages, all memory elements (i.e., flipflops, latches) should be made scannable such that their values can be controlled and observed.

2.4 BIST implementations

In order for BIST to be successful, BIST implementations have to meet the following requirements:

1. Low hardware overhead
2. Low test application time (i.e., low number of applied test patterns)
3. High enough fault coverage in order to meet the test quality requirements

Especially the first two points are hard to achieve at the same time. Several state-of-the-art BIST methods mainly focus on low hardware overhead [Lis87, Wai89, Hel92, Lem94], while other methods focus mainly on low test application times [Nag95, Duf91, Duf93, Vas93]. One of the main differences in all BIST implementations found in literature is the type of TPG that is used. Basically, TPGs can be divided into the following three groups:

1. **Exhaustive TPG (XTPG)**
2. **Pseudo random TPG (PRTPG)**
3. **Deterministic TPG (DTPG)**

Each of these groups of TPGs have their own advantages/disadvantages with respect to hardware overhead, test application time and fault coverage. This will be described in Subsections 2.4.1 to 2.4.3.

2.4.1 Exhaustive test pattern generator (XTPG)

The (theoretical) XTPG generates *all* possible test patterns. If a circuit has n PIs, the XTPG will generate 2^n patterns[1]. The XTPG generates all possible patterns and hence will also detect all possible *detectable* single SAFs; the fault coverage will be high. However, one can imagine that this large number of test patterns that have to be generated will not be feasible for large designs. Consider ISCAS'85 circuit c7552, see Appendix A. This circuit has 207 inputs (both PI and SFF inputs). An XTPG would have to generate 2^{207} patterns! Therefore, the XTPG is not practical for nowadays complex industrial designs as they often have far more than 207 inputs. In general one can say that the XTPG has a

+ low hardware overhead

– very large test application time

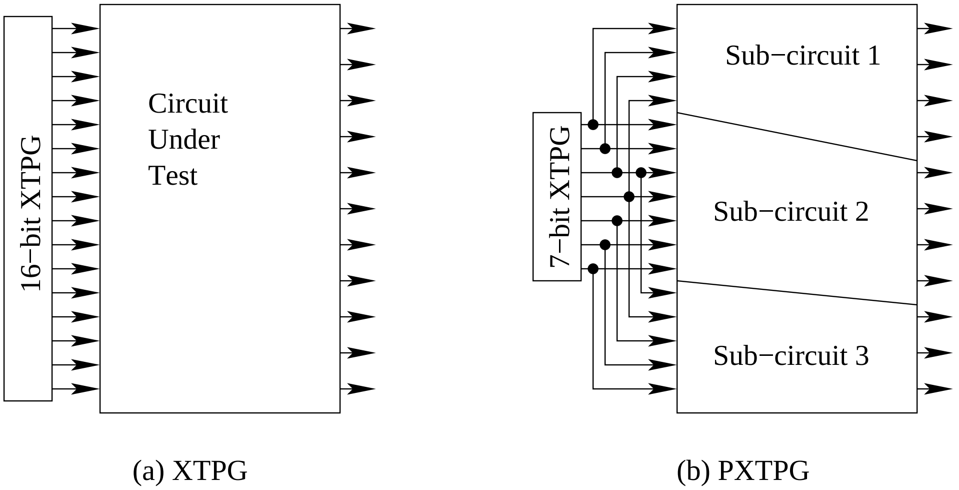
+ complete fault coverage (all detectable faults are covered)

The large number of test patterns of the XTPG can be reduced considerably when a *pseudo exhaustive test pattern generator (PXTPG)* is used. In this case the circuit is first partitioned into several independent sub-circuits. Each sub-circuit is tested with an XTPG. Because the number of PIs for each sub-circuit is much lower than the total number of PIs, the total number of test patterns that have to be generated can be reduced considerably.

Fig. 2.7(a) shows an example circuit with 16 PIs. The 16-bit XTPG will generate 2^{16} = 65536 patterns. Fig. 2.7(b) shows the same circuit in case PXTPG-testing is used. The circuit of Fig. 2.7(a) has been divided into three independent sub-circuits with respectively 4, 7, and 5 PIs. Each of these sub-circuits can be tested with an XTPG. Because the

---

[1] Or $2^n - 1$ patterns, because often the all-zero pattern will not be applied



(a) XTPG (b) PXTPG

Figure 2.7: (Pseudo) exhaustive test pattern generators

sub-circuits are independent, they can be tested in parallel by using the same XTPG with as width the number of inputs of the sub-circuit with the largest number of PIs. In the example, this sub-circuit has 7 inputs and hence $2^7 = 128$ test patterns will be generated, significantly less than the initial 65536 test patterns.

Although the number of test patterns that will be generated by a PXTPG is already reduced enormously, the PXTPG is still not very practical. Partitioning the circuit into independent sub-circuits is a complex problem by itself (*NP* complete) and is not always possible. Even when such sub-circuits are found, they are often still too large to be tested by a XTPG.

2.4.2 Pseudo random test pattern generator (PRTPG)

A PRTPG generates a set of PR test patterns which are applied to the CUT. These patterns are generated without taking into account any knowledge about the CUT structure. The patterns generated by a PRTPG are not really random, because they are determined deterministically by the implementation of the TPG. Although the patterns seem to be random, the PRTPG will generate the same sequence of patterns given the same initial pattern. Therefore this TPG is called a *pseudo* random test pattern generator.

Usually only a small subset of all possible test patterns are generated by the PRTPG. This way the test application time can be kept low, however at the cost of a lower fault coverage. Because the patterns are generated without any knowledge of the CUT, it can not always be guaranteed that all stuck-at faults are covered.

PRTPGs are often implemented by means of an LFSR or *Cellular Automaton (CA)* [Hor89]. Although an LFSR has very low hardware overhead, the patterns generated by the LFSR often fail to detect faults known to be *Random Pattern Resistant (RPR)*. RPR

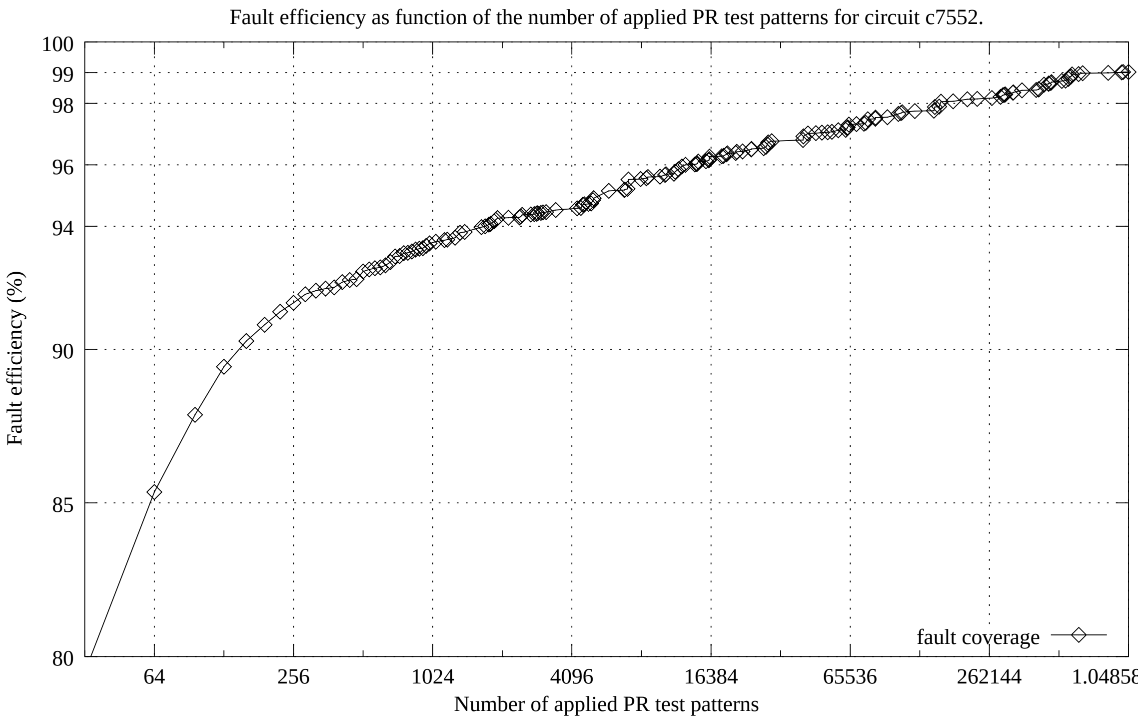


Figure 2.8: Fault efficiency versus number of applied PR patterns plot

faults are faults which can only be detected by a very small set of test patterns. Most faults, i.e., non RPR faults, can be detected by a relative large set of different test patterns. Because it is not feasible (with respect to test application time) to generate all test patterns, the number of test patterns generated by the PRTPG is often only a very small subset of all possible test patterns. Still it is likely that this subset contains at least one pattern with which a non RPR fault can be detected. Because there are only very few patterns which will detect an RPR fault, it is unlikely that this subset contains a pattern with which an RPR fault will be detected. Hence, the RPR faults remain undetected, reducing the fault coverage. Again one could increase the number of generated test patterns, but this will result in unacceptable long test application times.

Fig. 2.8 shows a plot of a PRTPG run on ISCAS’85 circuit c7552. The figure shows the *fault efficiency* as function of the number of applied PR test patterns. Note that the x-axis is in log2-scale! In Chapter 1, the fault coverage has been defined as the ratio of the number of actual detected faults and the *total* number of faults in the IC; the fault efficiency, on the other hand, is defined as the ratio of the number of actual detected faults and the number of *detectable* faults in the IC, i.e., the fault efficiency does not take into account faults that cannot be detected.

Fig. 2.8 shows that less than 200 PR patterns are required to reach 90% fault efficiency. Also 94% fault efficiency has been reached with less than 2000 PR test patterns.

But increasing the fault efficiency from 94% to 96% becomes already harder, although the number of required patterns is still acceptable in most cases; after 15,000 patterns the 96% level has been reached. But the remaining 4% becomes a problem. The 98% fault efficiency level already requires 180,000 patterns, more than 10 times as much as the 96% level. And even after applying 1 million PR test patterns, the 99% fault efficiency level has not been reached. This means that even for such a small circuit as the c7552 (only 3512 gates), over 1% of all *detectable* faults cannot be detected after applying one million PR patterns. In general one can say that the PRTPG has a

+ low hardware overhead

- large test application time

- low fault coverage

2.4.3 Deterministic test pattern generator (DTPG)

Both the XTPG and the PRTPG generate a lot of test patterns which are useless, these test patterns do not detect any ’new’ faults that were not already detected by other patterns. A deterministic TPG generates a set of deterministic test patterns, often generated by an ATPG tool. These deterministic test patterns do contribute to detecting ’new’ faults. By generating deterministic patterns, often the number of test patterns that have to be applied is low (short test application times), while still high fault coverages can be reached. The deterministic test patterns will contain the patterns that detect RPR faults.

However, the deterministic test patterns have to be stored in some way in the TPG. The state-of-the art deterministic BIST methods [Nag95, Duf91, Duf93] handle this in various ways as described in the following section. One of the most simple implementations would be using an embedded *Read-Only Memory (ROM)*. But for a circuit with many PIs and/or test patterns, the size of the ROM and hence the hardware overhead becomes unpractical large. For example, ISCAS’85 circuit c7552 with 207 inputs requires 128 ATPG test patterns (see Appendix A) for 100% fault efficiency. Hence a 26496 bits ROM would be required for a circuit with only 3512 gates, this would be an enormous overhead. In general one can say that a DTPG has a

– high hardware overhead

+ low test application time

+ high achievable fault coverage

2.4.4 BIST facilitation techniques

Fig. 2.9 shows a schematic test time versus hardware plot for several BIST methods. The ROM acting as DTPG and the LFSR acting as XTPG are put in this plot as extremes.

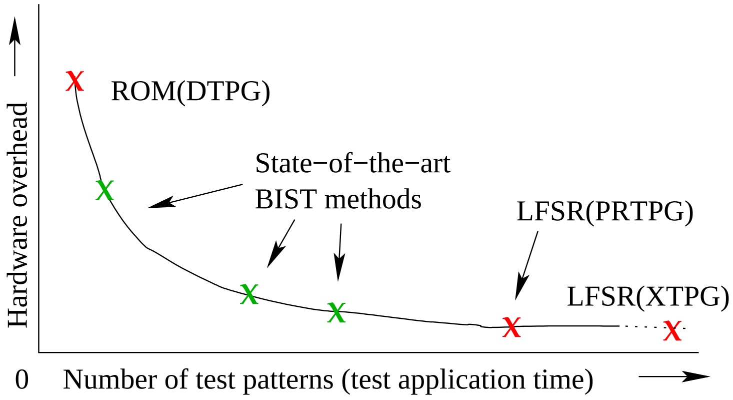


Figure 2.9: Test time versus hardware overhead plot for BIST

The XTPG has a very low hardware overhead, but an extremely long test application time, while the ROM with embedded ATPG test patterns has a short test application time, but at the cost of a significant hardware overhead. The PRTPG has the same low overhead as the XTPG, both implemented by means of an LFSR, but a much lower test application time. However the lower fault coverages of the PRTPG often do not meet the test quality requirements of the semiconductor industry. Basically, the techniques that are used to reduce the BIST overhead and/or to reduce the test application time and/or to increase the achievable fault coverages can be divided in techniques that:

1. *Modify the test*
   The CUT remains untouched, while the BIST techniques

   a. leave the TPG untouched (i.e., a low overhead LFSR), but modify the test patterns that will be applied to the CUT. Generated test patterns are modified, such that 'useless' patterns, i.e., patterns that do not detect new faults, become useful, i.e., they detect new faults. BIST methods based on this technique are described in Section 2.5.

   b. create/design a TPG that generates or embeds a full set of deterministic patterns, while trying to keep the hardware overhead low. BIST methods based on this technique are described in Section 2.6.

   c. create/design a TPG that does not embed a full set of deterministic test patterns, but only embeds parts of deterministic test patterns for detecting the hard-to-detect faults. BIST methods based on this technique are described in Section 2.7.

2. *Modify the circuit*
   The TPG remains untouched but the CUT is adjusted such that it becomes better (PR) testable. Mutation of the circuit is described in Section 2.8.

2.5 BIST facilitation techniques: Modify the test patterns

This section describes three BIST techniques, which modify the test patterns generated by a PRTPG in such a way that the fault-coverage will be increased. They are described in Subsections 2.5.1-2.5.3.

2.5.1 Reseeding of the LFSR

The LFSR starts generating patterns, given the seed of the LFSR. An *R*-bit LFSR with a primitive ChPol will generate all possible *R*-bit patterns, hence also the patterns that detect RPR faults. However, it might take a while before the LFSR generates a test pattern for a specific RPR fault. But with a right chosen seed, several RPR faults can be detected with only a few patterns, e.g., using a test pattern as seed that detects very hard RPR faults.

[Lem94] describes a method based on discrete logarithms [Poh78], which selects the ChPol and the seed of the LFSR which results in the smallest test sequence, detecting a given set of RPR faults. However, for circuits with many RPR faults, still a very large test sequence is necessary to cover most, if not all, RPR faults.

Instead of using one seed, *reseeding* can be used to make sure that most of the RPR faults are covered by the test sequence. After a first set of test patterns is generated using an initial seed, the LFSR is re-initialized with a new seed. This way a sequence of useless test patterns can be skipped, such that the LFSR will continue generating test patterns that do detect (RPR) faults.

[Hel92] describes a method which uses a *partially reconfigurable* LFSR and reseeding

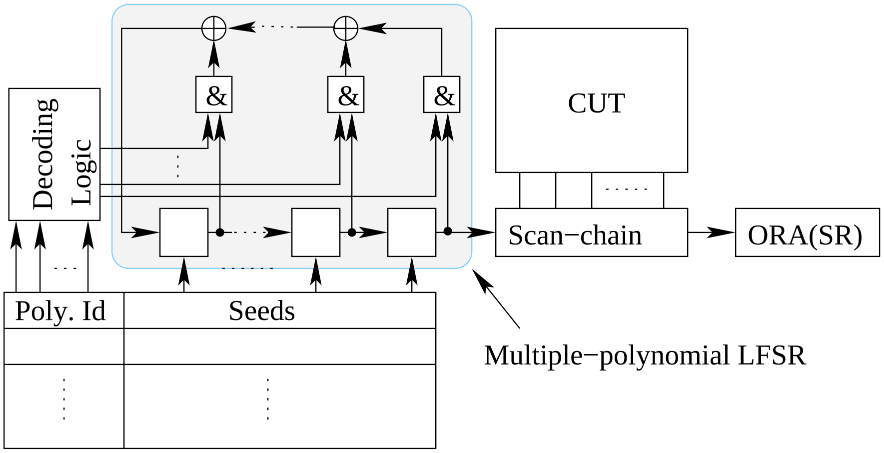
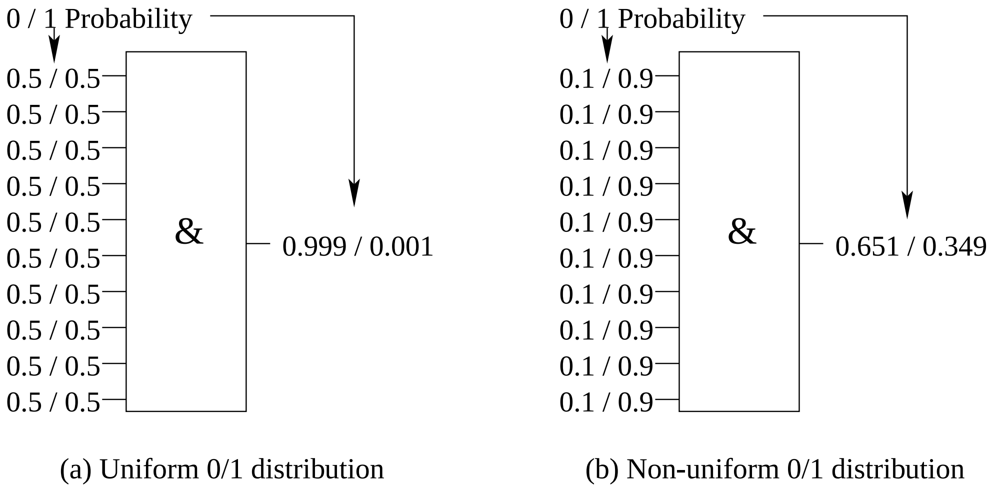


Figure 2.10: BIST scheme using Multiple-polynomial LFSR with reseeding



(a) Uniform 0/1 distribution (b) Non-uniform 0/1 distribution

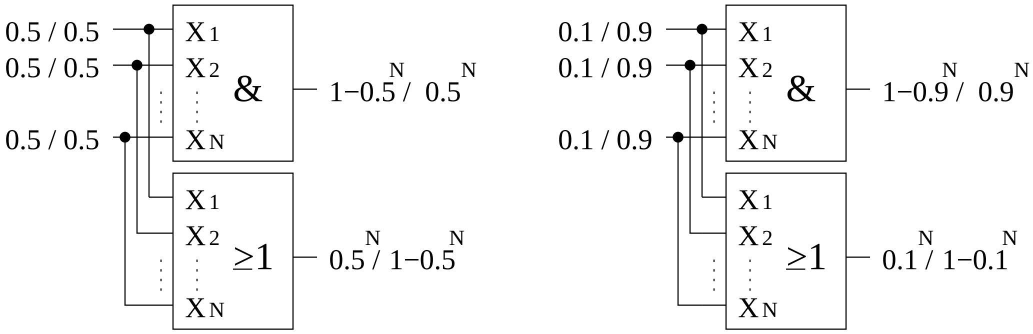
Figure 2.11: Impact input-value distribution on testability of a CUT

techniques to reduce the number of test patterns while keeping a high fault coverage. Partial reconfigurable means that the LFSR embeds multiple primitive ChPols and the LFSR can operate according to one of these ChPols. This BIST scheme is depicted in Fig. 2.10. Given the used ChPol (by the Polynomial Identifier (*Poly. Id*)) and the corresponding seed, the *Decoding Logic* enables/disables feedback loops through the AND-gates. The ChPol and seeds are chosen in such a way that test patterns for hard-to-detect faults will be generated. The PR bits generated by the out-tapping LFSR are shifted into the scan-chain of the CUT. The response of the CUT will be shifted out through the scan-chain into the ORA, implemented as a *Signature Register* [Bar87]. [Hel92] shows that this BIST scheme reduces the linear dependences between the bits shifted in the scan-chain. However, it is also mentioned that applications of this scheme are seen in the area of PXTPG, which means that the test application times still tend to be large for nowadays large and complex circuit designs.

2.5.2 Weighted Random TPG

When an LFSR is used as TPG, especially in case the LFSR has a primitive ChPol, the PI values will be uniformly distributed, i.e., each input will have a probability of 0.5 to be assigned a 0 or a 1. However, there are faults in the circuits which require a non-uniform distribution of 0 and 1 on the PIs in order to have a reasonable high probability to become detected by a PR pattern. This is illustrated in Fig. 2.11.

If all inputs of the 10-input AND-gate in Fig. 2.11(a) have an equal probability to be assigned 0 or 1, the probability that a SA0 fault at the output of the AND-gate will be



(a) Uniform 0/1 distribution (b) Non-uniform 0/1 distribution

Figure 2.12: Changing input-value distribution has positive impact on several faults and negative impact on other faults

detected[2] is very low (0.5^{10}). However, when these inputs have a much higher probability on a 1 than on a 0, the probability that the SA0 fault at the output will be detected, is much higher, as illustrated in Fig. 2.11(b).

Such a set of input values with a non-uniform distribution is called a *weight set*. Several faults which are RPR in case of a uniform input distribution, become much more testable with a certain weight set. However, this weight set can make one set of RPR faults better testable, but another group of faults worse testable. This is illustrated in Fig. 2.12. By increasing the probability of a 1 on the inputs of the gates, the probability that the SA0 fault at the output of the AND-gate will be detected, increases significantly, however at the same time, the probability of detecting the SA1 fault at the output of the OR-gate is reduced.

In order to be able to improve the detectability of all RPR faults in the CUT, multiple weight sets are required. In this case, the applied test set will be divided into sub test-sets. The first group of test patterns will be applied using weight set 1, the following group of test patterns will be applied using weight set 2, etc.. To apply weighted input values to the CUT, Boolean logic is required which converts uniformly distributed LFSR outputs into weighted inputs values for the CUT. E.g., in order to get a weight of 0.25 on a 1, a 2-input AND-gate can be used. The inputs of the AND-gate are connected to the LFSR and will have a probability of 0.5 on a 1. The output of the AND-gate will have a probability of 0.75 on a 0 and a probability of 0.25 on a 1.

The weighted random methods differ in the way they determine the weights. Several methods use testability analysis measures and heuristics to determine the weights [Lis87, Wai89], other methods use the input-value distribution of deterministic, i.e., ATPG, generated test sets [Mur90, Pom93], and again other methods are a combination of testability

---

[2] a SA0 fault at a line can only be detected, when that line is 1 in the fault-free case

Table 2.1: Weighted random benchmark results of several ISCAS'85 circuits

Circuit	#Wt. sets	#WR pat.	Circuit	#Wt. sets	#WR pat.
c880	2	1280	c3540	4	3840
c1355	3	2098	c5315	2	2048
c1908	6	5376	c6288	1	512
c2670	8	5888	c7552	10	9278

analysis and deterministic test distribution [Ree96].

Table 2.1, taken from [Wai89], shows weighted random benchmark results. Column *Circuit* shows the circuit name, Column *#Wt. sets* the number of weight sets that were necessary to detect all faults in the circuit, and column *#WR pat.* the number of weighted random patterns necessary to test all faults. [Wai89] only uses weights 1/16, 1/2 and 15/16 for input values. As could be expected, these results show that circuits which are known to be RPR, e.g., circuits c2670 and c7552, require more weight sets than more random susceptible circuits like the c880, c1355, c5315 and c6288. The number of weight sets for circuit c2670 and c7552 is quite large to be implemented for such small circuits (only 1193, respectively 3512 gates).

Also other weighted random results have shown that often a large number of different weight sets are required in order to achieve a high enough fault coverage within a reasonable test length. The logic overhead, necessary to enable the different weight sets, can be quite large. This overhead can be reduced by decreasing the number of weight sets and by limiting the number of allowed weights.[3]. However, this will decrease the fault coverage and increase the test length.

2.5.3 Mapping logic to replace useless patterns

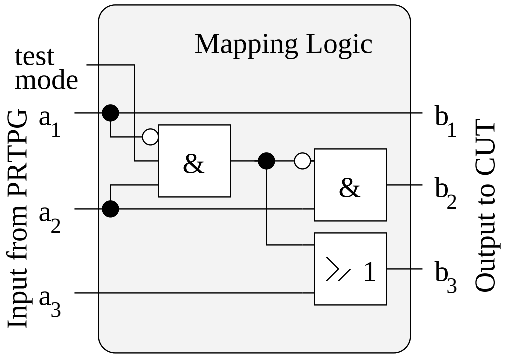
LFSRs/CAs used as PRTPG generate lots of test patterns that do not detect faults which were not already detected by previous test patterns. *Mapping logic* is logic which converts these 'useless' patterns into patterns which do detect faults that were not detected previously. Converting useless test patterns into patterns that detect RPR faults would be especially interesting. [Tou95] introduces such a method, which is based on *cube mapping*.

Test cube: A w-bit test cube Cu is a test pattern for which not all w bits necessarily have a specified value, i.e., $Cu = (c_1, c_2, \ldots, c_w) \in \{0,1,x\}^w$ with x representing **don't care**.

A test pattern $A = (a_1, a_2, \ldots, a_w) \in \{0,1\}^w$ is **contained** in a test cube $Cu = (c_0, c_1, \ldots, c_w) \in \{0,1,x\}^w$ if $\forall j, 1 \leq j \leq w | (a_j = c_j$ or $c_j = x)$.

---

[3] A weight of 0.231 is much more expensive in logic than a weight of 0.25 (2-input AND-gate)



(a) Cube mapping logic

a_1	a_2	a_3		b_1	b_2	b_3
0	0	0		0	0	0
0	0	1		0	0	1
0	1	0	→	0	0	1
0	1	1	→	0	0	1
1	0	0		1	0	0
1	0	1		1	0	1
1	1	0		1	1	0
1	1	1		1	1	1

(b) Mapped patterns

Figure 2.13: Cube mapping with source cube sCu={0 1 X} and image cube iCu={X 0 1}

The **Cube Mapping** $CM : \{0,1\}^w \to \{0,1\}^w$ is defined as follows: Given the source cube $sCu = (s_0, s_1, \ldots, s_w)$ and image cube $iCu = (i_0, i_1, \ldots, i_w)$, then the Cube Mapping $CM_{sCu \to iCu}(A) = B = (b_0, b_1, \ldots, b_w) \in \{0,1\}^w$ where if test pattern A is contained in *sCu*, then if $i_j = x$ then $b_j = a_j$ else $b_j = i_j$. If test pattern A is not contained in *sCu*, then $b_j = a_j \forall j$. In other words, when pattern A is contained in *sCu*, pattern A is transformed into a pattern which is contained in *iCu*. When A is not contained in *sCu*, A stays the same.

Given a required fault coverage level and given a maximum test length, the method described in [Tou95] tries to determine the mapping logic (based on cube mapping) required to reach this fault coverage level. The steps applied during this method are:

1. Do fault simulation on the set of test patterns generated by the PRTPG (LFSR).
2. Evaluate the fault coverage and identify the set of undetected faults
3. If the fault coverage is high enough, the process is finished.
4. If not, add a cube mapping.
5. Compute the transformed pattern set and do fault simulation on this test pattern set and go to step 2.

Fig. 2.13(a) shows an example of mapping logic based on cube mapping. In normal application mode (*test mode = 0*), the values on $b_{\{1,2,3\}}$ that will feed the CUT will be the same as on $a_{\{1,2,3\}}$ applied by the PRTPG. In test mode (*test mode = 1*), the values a applied by the PRTPG are mapped according to the cube mapping $CM_{\{01X\} \to \{X01\}}$. The corresponding mapping logic is shown in Fig. 2.13(a) and the corresponding pattern mapping is shown in Fig. 2.13(b).

The key problem that this method tries to solve is how to find the right cube mappings.

The described method in [Tou95] is based on solving a *binate covering problem* [Bra89]. Although good heuristics exist, the CPU requirements for solving the NP-complete binate covering problem still make this method infeasible for large industrial circuits.

2.5.4 Comparison of test pattern mutation techniques

Table 2.2 shows experimental results of weighted random testing and cube mapping. The table also contains experimental results of the fixed-biased BIST method [Als94], which is described further on in Subsection 2.7.2. Results are shown for several ISCAS benchmark circuits [Brg85, Brg89] and are taken from [Tou95].

Column *Circuit* in Table 2.2 shows the circuit name. Columns *Gates* and *FFs* show the number of gates, respectively the the number of FFs in the circuit; while Column *Random length* shows the number of required PR test patterns in order to reach 100% fault coverage of all detectable faults, when only an LFSR is used to generate the patterns. Columns *3-Weight*, *Fixed-biased* and *Cube mapping* show the results for respectively the weighted random method proposed by Pomeranz and Reddy [Pom93], the fix-biased method proposed by Alshaibi and Kime [Als94], and the cube mapping method proposed by Touba and McCluskey[Tou95]. For each of these methods, the hardware overhead is shown that corresponds to reaching 100% fault coverage for the given test length *Len*. This hardware overhead is split into added FFs (Column *FFs*) and Gate Equivalents (Column *GEs*), using the method suggested in [Har93]: [4] $0.5n$ GEs for an n-input NAND/NOR, $2.5(n-1)$ GEs for an n-input XOR, and 1.5 GEs for a 2-to-1 multiplexer.

These results show that the extra hardware overhead inserted to reduce the test length can be enormous and be more than the size of the circuit itself. Using the weighted random method of Pomeranz, the overhead can be over 100% as for circuit c2670, the

---

[4] One GE equals the size of a two-input NAND gate

Table 2.2: Comparison of Test Length and Required Hardware

				3-Weight			Fixed-biased			Cube mapping		
Circuit	Gates	FFs	Random length	Len	FFs	GEs	Len	FFs	GEs	Len	FFs	GEs
s420	196	16	1.1M	N/A	N/A	N/A	5K	18	>7	500	0	48
										1K	0	31
s641	379	19	1.0M	N/A	N/A	N/A	19K	20	> 21	500	0	23
										10K	0	12
s838	390	32	>100M	N/A	N/A	N/A	86K	19	>14	850	0	99
										10K	0	59
c2670	1193	0	4.6M	19K	5	1507	19K	54	>259	1K	0	260
				30K	5	1316				7K	0	114
c7552	3512	0	>100M	47K	6	3003	191K	111	>658	10K	0	214
				72K	6	2475				50K	0	183

overhead in GEs is 1316 or 1507 GE, depending on the test length, which are both more than the 1193 gates the circuit itself consists of. In case cube mapping is used, the hardware overhead is much less, but still often over 5%, which is already considered too much by most manufacturers. 114 GEs for circuit c2670 is still 9.5% overhead.

2.6 BIST facilitation techniques: Fully embedded deterministic test patterns

The following methods modify the TPG/LFSR in such a way that the TPG/LFSR also generates a full set of deterministic test patterns. The deterministic test set (e.g., generated by an ATPG) contains also patterns for the RPR faults such that the fault coverages achievable with BIST can be improved. Subsections 2.6.1, 2.6.2 and 2.6.3 describe respectively the pre-stored test BIST method, the LFSROM BIST architecture and the deterministic LFSR.

2.6.1 The pre-stored test BIST method

The pre-stored test BIST method [Nag95] uses the response of the CUT to generate the next test pattern. The TPG is implemented as a *Finite-State-Machine (FSM)*. The TPG runs through a sequence of states and each state represents a test pattern. The next state (hence, next test pattern) depends on the response of the CUT. If the CUT has a correct response, the TPG will come in the next valid state and will apply the pattern to the CUT, else the TPG will come in a different state and hence will apply a different pattern to the

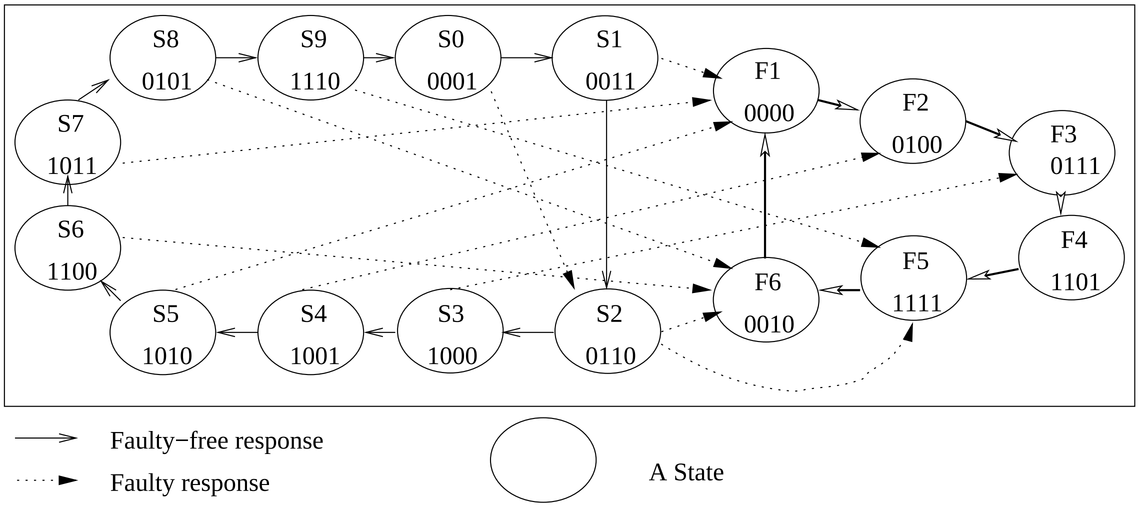


Figure 2.14: The state-sequence of the Pre-stored Test BIST method

Table 2.3: Hardware overhead and Fault coverage of a Pre-stored BIST

Circuit	Gates	Test patterns	Fault coverage	Hardware overhead
74283	35	11	100%	15%
74181	61	17	100%	7%
74280	55	11	100%	4%
74885	87	48	100%	27%
c432	160	34	99%	11%
c499	202	43	100%	18%
c880	383	51	99%	9%
c1355	546	85	100%	10%
c1908	880	119	100%	9%

CUT. A graphical representation of an example FSM is depicted in Fig. 2.14.

If all responses are correct and given that the FSM will start in state *S0*, the FSM will run through the sequence of states *S0*, *S1*, . . ., *S9* to *S0*. If an error occurs somewhere in this sequence, the TPG will follow another sequence of states and will return in a different end state after 9 applied test patterns. The states *F1*. . .*F6* can only be reached when an error occurred during the test and are considered fault states. So, without errors, the FSM will end in the state in which it started and one only has to check if this end state equals the initial state. Nagvajara [Nag95] describes how the FSM should be designed such that it has a low probability of aliasing. In this case aliasing is the effect that a faulty circuit ends in state *S0*, because then it would falsely be considered a fault-free circuit.

Table 2.3 shows experimental results for the pre-stored test, taken from [Nag95]. Only results for some very small circuits are shown. Columns *Circuit* and *Gates* show the name of the circuit respectively the number of gates in the circuit. Column *Test Patterns* shows the number of ATPG test patterns needed to reach the fault coverage shown in Column *Fault Coverage*. Column *Hardware overhead* shows the hardware overhead (percentage) of the FSM.

The hardware overhead of this BIST system is linear with the number of pre-stored test patterns and the pattern size. Therefore, this method is only feasible for circuits with a very compact ATPG test set which can be pre-stored. Otherwise, although often 100% fault coverage is achieved, the hardware overhead will be far over 5% and will be often considered too high by IC manufacturers. In Table 2.3 only circuit 74280 results in less than 5% overhead, all other circuits result in more overhead; up to 27% for circuit 74885.

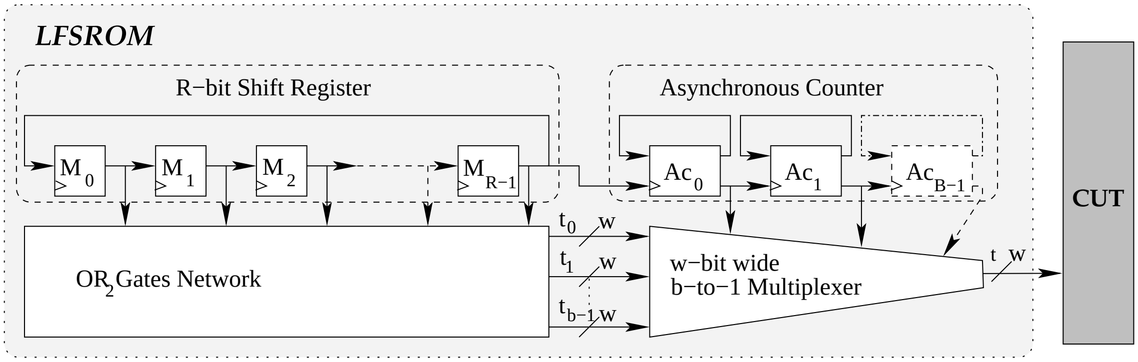


Figure 2.15: The LFSROM architecture

2.6.2 The LFSROM architecture

The LFSROM architecture, presented in [Duf93], uses a shift register in combination with an OR_2 gates (2-input OR gates) network to store a deterministic test set; the LFSROM stores a deterministic test set containing T test patterns of w bits width. It is composed of the following four parts, depicted in Fig. 2.15:

1. A *R-bit shift register*
   The R-bit shift-register shifts through the states from $[100\cdots0]$, $[010\cdots0]$ to $[000\cdots1]$ with a cyclic period R.

2. An *OR_2 gates network*
   The R states of the shift register are transformed into R deterministic test patterns by the OR_2 gates network. These test patterns are w bits wide.

3. A *w-bit wide b-to-1 multiplexer*
   Instead of generating R test patterns, the test set contains T deterministic test patterns that have to be generated. Multiple subsets of test patterns can be generated by the OR_2 gates network in parallel. The w-bit wide multiplexer can be used to select one of the $b = \lceil T/R \rceil$ subsets of test patterns.

4. An *Asynchronous Counter*
   An $B = \lceil log_2(b) \rceil$ bit asynchronous counter is used control the multiplexer and to select which of the b subsets of test patterns will be applied to the CUT.

Table 2.4 shows experimental results for the LFSROM architecture, taken from [Duf93]. The overhead of the LFSROM architecture is compared to the overhead of using a ROM to store the test patterns. Column *Circuit* shows the circuit name, Columns *T* and *w* show respectively the number of test patterns to store and the width of the test patterns. Column *R* shows the length of the R-bit shift register and Column *b* shows the value for b of the b-to-1 multiplexer. Columns *ROM* mm^2 and *LFSROM* mm^2 show the silicon-size of the

Table 2.4: Hardware overhead and Fault coverage of the LFSROM

Circuit	T	w	R	OR_2	b	ROM mm^2	LFSROM mm^2
20x20 multiplier	41	49	25	69	2	1.14	0.54
ALU	86	67	25	133	4	1.62	1.23

ROM, respectively the LFSROM, necessary to store the test for these circuits.

Unfortunately, only the silicon-overhead of the LFSROM compared to a ROM is shown. The silicon sizes of the structures themselves is not listed. Although the overhead of the LFSROM is 52% (from 1.14mm^2 to 0.54mm^2) respectively 24% (from 1.62mm^2 to 1.23mm^2) less than that of a ROM, it is still in the same order of magnitude and therefor probably not feasible for circuits with large deterministic test sets.

2.6.3 The deterministic LFSR

In [Duf91] and [Vas93] a method is described which tries to design a deterministic LFSR that starts with generating a complete set of deterministic test patterns *T*, after which it will generate PR patterns. In order to achieve this, the characteristic matrix T_c of Fig. 2.4 is converted into the matrix T_s. This matrix loses the general in-tapping structure, but still has the same ChPol. So when T_c has a primitive polynomial, T_s also has a primitive polynomial and the corresponding LFSR will generate all possible patterns of length $R = w$. The ChPol of an LFSR can be calculated using Eq. 2.5, where *det* is the determinant and *IM* is the *Identity Matrix*. The elements of the main diagonal in the Identity Matrix are ones, all other elements are zero:

$$ChPol = det(T_c + IM \cdot x) \tag{2.5}$$

When the LFSR has a general in-tapping structure, the h_r values of Eq. 2.1 can be found in the last row of matrix T_c. In case of a general in-tapping LFSR, the next state of memory cell X_r only depends on the current state of cells X_{r-1} and/or X_{R-1}.

Consider the non-singular *R*x*R* matrix *A*. The pattern *X(t)* can be mapped into pattern *X'(t)* by applying Eq. 2.6.

$$X'(t) = X(t) \cdot A \tag{2.6}$$

An *R*x*R* matrix is non-singular when the matrix has *R* independent rows and columns. The matrix A should be non-singular, otherwise the inverse matrix A^{-1} does not exist. This inverse matrix is necessary to calculate the matrix T_s. Eq. 2.6 can be rewritten to:

$$X'(t) \cdot A^{-1} = X(t) \cdot A \cdot A^{-1} \Leftrightarrow X'(t) \cdot A^{-1} = X(t) \tag{2.7}$$

Combining Eqs. 2.2 and 2.7, we get:

$$\begin{aligned} X'(t+1)\cdot A^{-1} &= X'(t)\cdot A^{-1}\cdot T_c \\ X'(t+1) &= X'(t)\cdot A^{-1}\cdot T_c\cdot A \end{aligned}$$

and finally into:

$$X'(t+1) = X'(t)\cdot(A^{-1}\cdot T_c\cdot A) = X'(t)\cdot T_s \tag{2.8}$$

T_s and T_c are called similar matrices; these matrices possess the same ChPol [Duf91]: *det*$(T_c + I\cdot x)$ = *det*$(T_s + I\cdot x)$.

The LFSR corresponding to the matrix T_s can be found as follows: The next state of memory cell X_i can be found by taking the exclusive OR of the current states of the memory cells X_j for which element (j,i) (row j and column i) in matrix T_s is 1.

According to [Poh78], when the T_c is initialized with the state $X(0)$=[`100`$\cdots$`0`], independent of the ChPol, the LFSR will first generate the following state sequence:

$$\begin{aligned} X(t=0) &= [\texttt{100}\cdots\texttt{0}] \\ X(t=1) &= [\texttt{010}\cdots\texttt{0}] \\ X(t=2) &= [\texttt{001}\cdots\texttt{0}] \\ \vdots &= \vdots \\ X(t=N-1) &= [\texttt{000}\cdots\texttt{1}] \end{aligned}$$

When each of these states $X(t)$ is transformed into $X'(t)$, using Eq. 2.6, it is clear that each state $X'(t=i)$ equals to row i of matrix A. Hence, if the LFSR with characteristic matrix T_s is initialized with the first row of A, all following rows of A will be generated. After all rows of A have been generated, the LFSR continues with the PR states.

So if A would be the test set containing $T = R$ independent deterministic patterns, this method can be used to design an LFSR which will generate all test patterns of this test set. However, a drawback of this method is that the LFSR must have the same width (R) as the test patterns (w). For a circuit with many inputs (PIs and SFF outputs), this can be enormous. If the test set A is singular, extra bits or patterns have to be added to make it non-singular; this will make the LFSR even larger!

Fig. 2.16 shows an example of the deterministic LFSR for a deterministic test set containing 4 patterns. Given the seed `0101`, which is the first test pattern in test set A, the deterministic LFSR first generates the remaining patterns/states of test set A, after which it will generate two PR states before it will generate state `0101` again. The ChPol found in the last row of the T_c given in Fig. 2.16 is not primitive. T_s has the same non-primitive ChPol and therefore the deterministic LFSR will not generate all $2^{n-1} = 15$ possible states, but only 6 (4 deterministic, 2 PR) different states, given seed `0101`.

[Duf91, Vas93] do not show results with respect to the hardware overhead. It is assumed that all inputs are controlled through (S)FFs and therefore the LFSR memory cells will not result in much hardware overhead. However, the main hardware overhead consists of the overhead of the necessary XOR gates. In [Geu97a], experimental results for

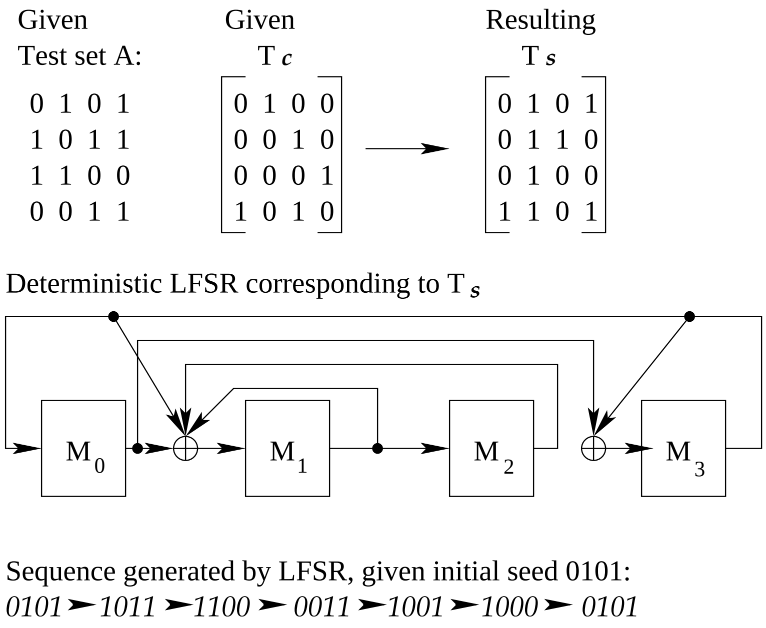


Figure 2.16: Deterministic test set embedded in an LFSR

the ISCAS'85 benchmark circuits [Brg85] are given of the deterministic LFSR, with respect to the number of necessary two-input XOR gates; see Table 2.5.

Column *Circuit* shows the circuit name, Column *#Patterns* shows the number of deterministic test patterns and Columns *#Inputs* and *#Gates* show respectively the number of

Table 2.5: Hardware overhead of the Deterministic LFSR

Circuit	#Patterns	#Inputs	#Gates	XOR_2
c17	6	5	6	5
c245	19	14	63	42
c432	45	36	160	306
c499	60	41	202	262
c880	27	60	383	313
c1355	96	41	546	546
c1908	127	33	880	766
c2670	76	233	1193	2916
c3540	130	50	1669	1226
c5315	129	178	2307	3708
c6288	33	32	2416	201
c7552	160	207	3512	5405

inputs and the number of gates of the circuit. Column XOR_2 lists the number of two-input XOR gates which are necessary to implement a deterministic LFSR which starts with generating all patterns of the deterministic test set. If necessary, extra patterns and or bits are added in order to get a non-singular NxN test set.

The experimental results in Table 2.5 show that the hardware overhead with respect to the number of XOR gates, compared to the number of gates in the circuit, is large; circuits c432, c399, c2670, c5315, 7552 require even more XOR gates than the original number of gates in the circuit. The remaining circuits, except circuit c6288, require approximately the same number of XOR gates compared to the number of gates in the circuit itself. This means that the deterministic LFSR-overhead for these circuits is approximately 100% or more[5]. Only circuit c6288 requires a much lower number of XOR gates compared to the number of gates in the circuit, 201 XOR gates compared to 2416 gates in the circuit itself, resulting in an overhead of 201/2416*100%=8.2% when only the XOR gate overhead is taking into account; i.e., ignoring the overhead for the 33 (the number of test patterns; $R = T$) memory cells of the deterministic LFSR. Both the number of PIs and the number of test patterns for circuit c6288 are relatively small compared to the number of gates in the circuit. As the size of the deterministic LFSR depends both on the number of PIs and the number of test patterns, the size of the deterministic LFSR is also kept relatively small.

The deterministic LFSR is only feasible for circuits with only a relative small number of inputs and a relatively small test set compared to the circuit size. Embedding complete deterministic test sets, or creating a deterministic LFSR for circuits with a large number of inputs, can result in an overhead of 100% or more, as shown in Table 2.5.

2.7 BIST facilitation techniques: Partially embedded deterministic test patterns

This section describes BIST techniques which do not embed a full set of deterministic test patterns, but only embed partially specified deterministic patterns for the hard-to-detect faults. Subsections 2.7.1, 2.7.2 and 2.7.3 describe respectively mapping logic, fixed-biased PR BIST and bit-flipping BIST.

2.7.1 Mapping logic

In Subsection 2.5.3 it has already been shown that mapping logic can be used to map ’useless’ patterns generated by the PRPG/LFSR into patterns that do detect hard-to-detect faults. Mapping logic can also be used to take advantage of redundant information that is present in deterministic test sets in order to reduce the TPG size and/or test length.

---

[5] It is assumed that an XOR gate results in the same overhead as an arbitrary gate in the circuit, which is in general not the case

In [Kag96] a BIST method is presented which uses a binary counter as TPG, instead of an LFSR. This TPG generates all test patterns of a given deterministic test set. This test set is represented as a T x w matrix A, T patterns w bits wide. The counter is initialized with pattern t_{low} and counts until pattern t_{high} is reached. The patterns t_{low} and t_{high} are chosen in such way that the *cyclic distance* between t_{low} and t_{high} is as small as possible, while still all patterns of A will be generated by the counter. The cyclic distance of A is the minimum of the *straight distance* and the *wraparound distance* of A. The straight distance of A is defined in Eq. 2.9, where t_i, $1 \leq i \leq T$, are the decimal values of the test patterns in A.

$$\text{Straight distance} = max\{t_i\} - min\{t_i\} \tag{2.9}$$

The wraparound distance is given in Eq. 2.10, where t_i,t_j are *consecutive* patterns in A, sorted on decimal value, and $t_i > t_j$.

$$\text{Wraparound distance} = 2^w - max\{t_i - t_j\} \tag{2.10}$$

The minimum cyclic distance is the minimum number of patterns that the counter has to generate such that all deterministic patterns of A will be generated.

[Kag96] introduces the following techniques in order to transform matrix A into A', with a smaller cyclic distance:

1. *Pre-processing operations*
   These operations reduce the width of the patterns of the test set. This is done by deleting columns from A. Mapping logic between the counter and the CUT is used to re-insert the removed bits/columns. These operations consist of:

   - *Constant column elimination*
     Columns with only ones or zeros are removed. These columns do not have to be generated by the counter.
   - *Identical column merging*
     Two columns c_i and c_j are identical if for every row position the value of column c_i equals the value of column c_j. Only one of these columns has to be generated by the counter.
   - *Complementary column merging*:
     Two columns c_i and c_j are complementary when for every row position the value of column c_i is the opposite of column c_j. Only one of these columns has to be generated by the counter, the other column can be retrieved by inverting this column.

2. *Basic operations*
   These operations adapt the to be generated patterns into other patterns such that the distance between these patterns is as low as possible. They do not delete columns, but adjust the matrix by inverting columns or interchange columns with other columns. The basic operations consist of:

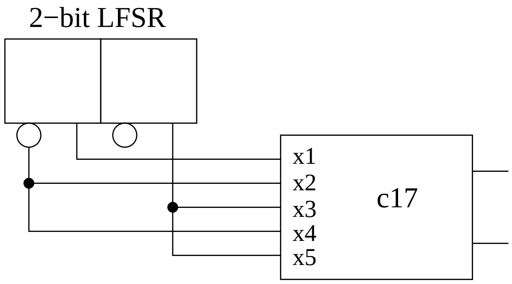
Table 2.6: Test length of Counter-based method compared to weighted random approaches

Circuit	Org. test set	WR.-1[Mur90]	WR.-2[Maj94]	Counter
c409	14x41	1,464,030	15,862	22,064
c432	6x36	14,971	1,619	125
c880	11x60	74	61	29
c1355	12x41	6,472,654,940	29,758,994	122,544,062
c1908	14x33	4,520	2,181	1,169
c3540	22x50	4,195,415	18,167	970
c5315	7x178	135,026,106	475,961	62
c6288	36x32	2,804,690,838	4,389,665	98,003,134

- Column permutation
  It is not necessary that column c_i has to be connected to PI c_i of the CUT. The columns of the matrix can be permuted without any consequences as long as the output of the right counter cell is connected to the right PI(s) of the CUT. This is called *column permutation*.
- Complementary column generation
  Instead of the column c_i itself, the counter can also produce its inverse $\overline{c_i}$. This is very useful when it would result in a shorter cyclic distance. This is called *complementary column generation*.

The mapping logic necessary to connect the counter output to the correct CUT inputs consists only of fan-outs and inverters.

Table 2.6 shows experimental results for eight ISCAS'85 benchmark circuits [Brg85] taken from [Kag96]. Column *Circuit* shows the name of the circuit and Column *Org. test set* shows the original test set size, i.e., <the number of test patterns> x <test pattern width>. Columns *WR.-1*, *WR.-2* and *Counter* show the number of PR patterns, i.e., the test length, for respectively the weighted random methods presented in [Mur90], [Maj94] and this counter-based method. For five circuits, i.e., circuits c4432, c880, c1908, c3540 and c5315, the counter-based method outperforms both weighted random approaches. The weighted random method of [Mur90] is even outperformed for all eight circuits. However, the embedded test set sizes are not very large and even for a very small circuit c1355 (only 546 gates), the test length still exceeds 10 million patterns for all approaches. [Kag96] also claims that this method becomes impractical when the Txw test set matrix contains more than 50 patterns, or a pattern width larger than 200. Test sets for nowadays large VLSI circuits are much larger, therefore this BIST method does not seem to be feasible for large industrial designs.



Test set

x1	x2	x3	x4	x5
1	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0

Figure 2.17: LFSR input reduction for ISCAS'85 circuit c17

In [Che95a] also techniques are introduced to reduce the width of the LFSR that is used to drive the PIs of the CUT. In this case no deterministic test patterns will be embedded; the idea is to reduce the width of the LFSR and hence the test length. This is achieved by shorting *Compatible* inputs and *Inversely compatible* inputs. Compatible inputs are inputs which can be shorted together without introducing untestable faults. Inversely compatible inputs are inputs that can be shorted together by means of an inverter without introducing untestable faults. Fig. 2.17 shows the minimal LFSR with which ISCAS'85 benchmark circuit c17 [Brg85] can be tested. With only a 2-bit LFSR, all faults in circuit c17 are detected.

Table 2.7 shows experimental results taken from [Che95a] for several ISCAS'85 [Brg85] and ISCAS'89 [Brg89] circuits. Column *Circuit* lists the circuit name and Column *#Inputs* shows the number of inputs (PIs and SFF outputs). The number of stages of the LFSR are shown in Column *#LFSR stages*.

The results listed in Table 2.7 show that the number of LFSR stages can be reduced with a factor of 2 or even 3. For many circuits, a maximum length pattern sequence can be generated by the LFSR ($2^R - 1$ patterns, where R is the number of LFSR stages), while still having a test length below the 1 million patterns (R=20). Only circuits c2670 (R=22), s838 (R=35) and s15850 (R=31) result in a maximum length sequence exceeding the 1 million patterns. If a maximum length sequence is not considered feasible for a circuit, e.g., circuits s838 and s15850, still only a part of the LFSR sequence can be applied during test, just as in the case an LFSR is used without input reduction at the cost of a lower fault coverage. The input reduction method seems to be rather complex and is CPU time consuming (no CPU time information available) which can be a bottleneck for large industrial designs.

2.7.2 Fixed-biased PR BIST

The fixed-biased PR BIST architecture, presented in [Als94] is using a mixture of fixed bits plus *biased PR patterns*. In this case biased PR patterns are patterns in which all bits have the same probability p, $0 < p < 1$, of being 1. The basic structure of this

Table 2.7: Input reduction results for several ISCAS'85 and ISCAS'89 circuits

ISCAS'85			ISCAS'89		
Circuit	#Inputs	#LFSR stages	Circuit	#Inputs	#LFSR stages
c17	5	2	s420	35	19
c432	36	12	s641	54	17
c1355	41	11	s838	67	35
c1908	33	13	s1423	91	13
c2670	233	22	s15850	611	31

BIST architecture is depicted in Fig. 2.18.

The Pattern Generation and Control Logic (*PGC Logic*) drives and controls a scan-chain, consisting of the input FFs to the CUT. This scan-chain is referred to as the *idler register*. The idler register applies test patterns to the CUT in test mode; the response of the CUT is captured into a MISR and the signature of the MISR and the end of the test is used to decide whether the circuit is good or not. This BIST method is based on the following two key ideas [Als94]:

1. *Bit-fixing*: Assign certain idler register bit positions to hold fixed values during a period of the test.

2. *Biased PR testing*: Applying biased PR test patterns to all the idler register bits not fixed at any given time.

A biased PR pattern is delivered for each shift of the idler register; it is a test-per-clock approach, which is faster than the more commonly used test-per-scan BIST approach. In a test-per-clock approach, a test pattern is applied every clock cycle, while in a test-per-scan approach first the test pattern has to be shifted in the scan-chains, which takes

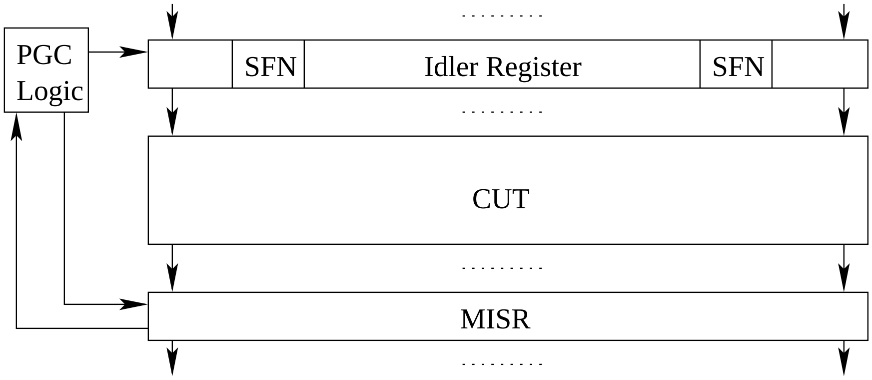


Figure 2.18: The fixed-biased BIST architecture

multiple clock-cycles (at least as many as the longest scan-chain). Bit-fixing is accomplished by special *Scan, Fix and Normal (SFN)* scan-cells. These scan-cells can operate in scan, *fixed-scan* and normal mode. The scan and normal mode are the same as in a regular scan-cell. In the fixed-scan mode, the cells stay fixed at 1 or 0, while the value of the previous cell is passed to the next cell in the chain, bypassing this fixed cell.

[Als94] describes the FBIST algorithm which is used to determine the bits that have to be fixed, the bias value and the number of different sequences (different fixed bits and/or different bias values) the test will consist of. The concept of this FBIST algorithm is summarized as follows:

1. Use PR test patterns to detect the easy-to-detect faults.
2. Use an ATPG tool to generate sparsely assigned test patterns for each of the remaining undetected faults.
3. Use a bit selection algorithm to determine which bits, if any, need to be fixed, given the set of generated ATPG patterns. Use a bias determination algorithm to calculate the appropriate bias value, and use fault simulation to evaluate the fault coverage of these fixed-biased PR test patterns. For the detected faults, remove the ATPG patterns from the set of sparsely assigned ATPG patterns.
4. If all faults are detected: Stop; Otherwise go to step 3 with the remaining ATPG patterns.

The bias is determined by the ratio between 1s and 0s in the sparsely assigned set of test patterns. A bit is permanently fixed at 1(0) when all of the ATPG patterns have a 1(0) at that position. A bit is temporary fixed at 1(0) when more than a given percentage of the ATPG patterns have a 1(0) at that bit position.

The hardware overhead of the fixed-biased method consists of the extra hardware costs of the SFN cells and the PGC logic. According to [Als94], the SFN cells are approximately 2.5 times the size of a normal scan-cell. It is therefore important that the number of bits that will be fixed is kept low. The PGC logic consists of:

- A ROM to store the configure sequences and the control sequences, i.e. the number of patterns used in a configure, the bias value, the bits to fix, etc.
- A PRTPG
- Combinational logic to generate biased PR bits from the PRTPG
- A control PLA and counters which control the test operation.

Table 2.8 shows experimental results of the fixed-biased BIST method with respect to the required test length to achieve 100% fault coverage, compared to other BIST methods [Als94]. Column Circuit shows the ISCAS'85 circuit name. Column Random T shows

Table 2.8: Comparison of test lengths of the fixed-biased method with other techniques

Circuit	Random	3-Weight[Pom93]			Fixed-biased[Als94]		
	T	T	Fix	Seq	T	Fix	Seq
c432	768	1024	0	1	432	0	0
c499	1216	2048	0	2	451	0	0
c880	7488	8192	3	8	5280	0	0
c1355	2816	4096	1	4	1599	0	0
c1908	14K	4096	0	4	5148	1	1
c2670	4M	58K	170	57	19K	37	4
c7552	$>$64M	151K	207	37	191K	94	6

the required number of PR test patterns (test length) needed to detect all faults. The following six columns show respectively the number of patterns T, the number of fixed bits Fix and the number of different fixed sequences Seq for respectively the 3-Weight PR BIST method presented in [Pom93] and the fixed-biased method of [Als94].

The first five circuits are random susceptible circuits, all circuits are testable with less than one million PR patterns. Although the test length can be reduced by the listed BIST methods, the reductions often do not justify the overhead of the BIST methods. The last two circuits, i.e., circuits c2670 and c7552, are RPR circuits and using one of the listed BIST methods can be necessary to reduce the test length to a more acceptable level, e.g., below 1 million patterns.

The BIST methods of [Pom93] and [Als94] have comparable results with respect to the test length (T) in Table 2.8. However the number of bits to fix and the number of sequences is smaller for the method presented by [Als94].

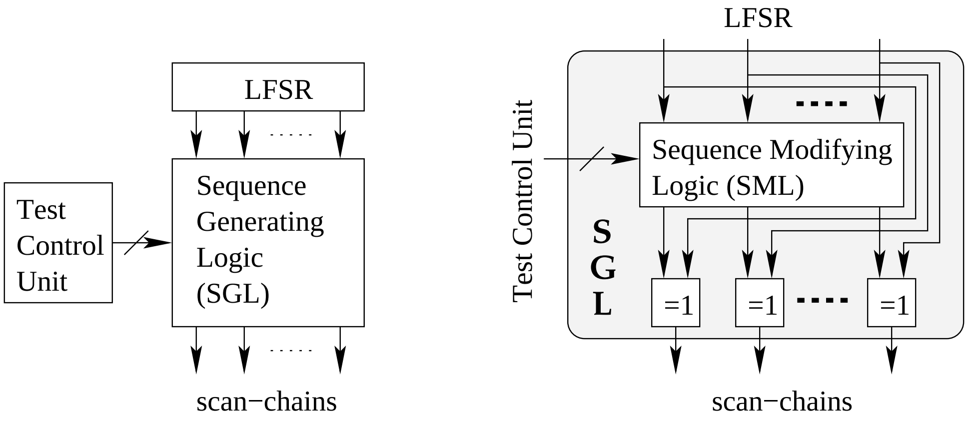
Although the fixed-biased method is not limited to small circuits (note that for larger circuits the overhead of the PGC logic becomes relatively small), this method does not seem to be practical for RPR circuits. If a circuit is RPR, often a lot of fixed-bits and sequences have to be used to reduce the test length. As a result, the overhead of the idler register due to the SFN cells and the size of the PGC logic, especially the ROM-size required to store all configurations, becomes unacceptable.

2.7.3 Bit-flipping BIST

In [Kie98] a BIST method is introduced which is based on flipping of bits generated by the LFSR. This BIST system is based on the following observations [Kie98]:

1. Given a set of PR patterns, deterministic patterns can easily be embedded by modifying just a very small number of bits [Wun96].

2. In a PR test set only a very small number of patterns contribute to the fault coverage



(a) Bit-flipping BIST structure (b) Sequence Generating Logic example

Figure 2.19: The Bit-flipping BIST architecture

and within these patterns only a few bits need to be specified.

3. Very often, deterministic test patterns can be clustered into a few sets such that all the patterns of a set look very similar [Pat91].

4. Every autonomous BIST method must contain a BIST control unit containing a bit counter and a pattern counter for generating the shift/capture signal and the "test-end"-signal.

The system is depicted in Fig. 2.19(a). It consists of an LFSR, the *Sequence Generating Logic (SGL)* block and the *Test Control Unit*. The SGL modifies the bit sequences generated by the LFSR at several bit positions and applies these modified sequences to the scan-chains. The structure of the SGL is depicted in Fig. 2.19(b). It consists of the *Sequence Modifying Logic (SML)* and XOR (=1) gates. The current state of the LFSR, the bit counter and the pattern counter of the Test Control Unit serve as input for the SML. There the SML can overcome dependencies between LFSR outputs, no phase-shifting is required. Even a single LFSR output can be used to drive the scan-chains. However, for some circuits this has a negative impact on the fault coverage and still multiple LFSR outputs should be used to drive the scan-chains.

The SML is constructed iteratively. The main steps of the algorithm are described in the following text. To reach an efficient implementation of the SML, the logic minimization routines EXPAND and REDUCE have been integrated [Bra84].

1. Check whether the current fault coverage is sufficient. If so, stop.

2. Compute the bits in the sequences that should be fixed. These are bits that should stay fixed because these bits are essential to detect some hard-to-detect faults. These fixed bits can be bits of deterministic patterns embedded during a previous iteration.

Table 2.9: Bit-flipping BIST silicon overhead

Circuit	FE random	Core area (μm^2)	Scan area	BIST area SGL	BIST area Other	BIST area Total
p4210	94.47%	113,877	12.05%	12.01%	10.58%	22.60%
p4250	96.09%	97,110	8.53%	8.92%	11.95%	20.87%
p8689	88.56%	217,053	10.37%	19.72%	5.69%	25.41%
p8873	96.40%	289,053	7.40%	6.12%	4.14%	10.27%
p13651	99.84%	308,727	8.48%	0.27%	3.91%	4.18%
p14473	86.39%	333,612	9.31%	30.27%	3.87%	34.15%
p27530	98.23%	542,394	3.32%	13.04%	2.32%	15.36%
p44177	97.25%	1,324,980	12.38%	3.47%	0.95%	4.42%
p52251	99.32%	1,482,687	14.19%	0.61%	0.88%	1.50%
p52922	92.66%	1,889,622	9.34%	7.57%	0.86%	8.42%
p64984	94.07%	1,714,725	14.71%	10.45%	0.70%	11.26%
p80590	99.09%	2,492,622	4.94%	2.33%	0.58%	2.91%

3. REDUCE SML
4. Embed deterministic patterns. An ATPG tool is used to generate a sparsely assigned test pattern.
5. EXPAND SML

The application of bit-flipping BIST on industrial designs is presented in [Kie00]. Table 2.9, taken from [Kie00] shows bit-flipping BIST results with respect to the silicon overhead for several Philips industrial designs. These results are obtained in case the goal was to achieve complete fault coverage with 10,000 PR patterns. Column *Circuit* shows the name of the circuit and Column *FE random* shows the achieved PR fault efficiency without bit-flipping. Column *Core area* and *Scan area* show the silicon area of the core and the percentage of area required for making the circuit fully scannable. Column *BIST area* shows the percentage of silicon area spent on the BIST logic, divided into the area of the SGL (*SGL*), the area of the remaining BIST logic, consisting of LFSR/MISR and test control logic (*Other*), and the total percentage of silicon area spent on BIST (*Total*).

The results in Table 2.9 show that the area overhead for making the circuit fully scannable is in general 5%-15%, except for circuit p27530. The percentage of area spent on the SGL differs a lot between different circuits. The size of the SGL primarily depends on the random testability of the circuit, as claimed in [Kie00]; RPR circuits, i.e., circuits p8689, p14473, p52922, require larger SGL than more random susceptible circuits, i.e., circuits p13651, p52251, p80590. As the LFSR, MISR and BIST test logic hardly increase with circuit size, the percentage of silicon area spent on these logic blocks decreases with circuit size. For the larger circuits the SGL dominates the BIST silicon overhead, as is shown in Column *Total* in Table 2.9. For circuits that are well random testable, the BIST overhead is limited, i.e., smaller than 5%, but for RPR circuits, the overhead can become larger than 10% up to 34% for circuit p14473. This overhead can be reduced by lowering the required fault coverage or applying more PR patterns, as shown in [Kie00].

2.8 BIST facilitation techniques: circuit mutation

The previous sections have shown that many state-of-the-art BIST methods exist that improve the BIST fault coverage. However, the hardware overhead of these BIST methods increases for circuits that are more RPR (except for the BIST methods in Section 2.6, their hardware overhead depends on the size of the deterministic test set they embed). For large RPR industrial circuits, the state-of-the-art BIST methods become impractical due to too much hardware overhead, too low fault coverages and/or too large CPU consumption by the BIST method's algorithm. But when the circuits are better PR testable, the state-of-the-art BIST methods are often more promising in reaching high fault coverages with low overhead. If the circuits even become very PR testable, they can even be tested with an LFSR without the necessity of complex BIST implementations. This section describes how the circuit can become better PR testable by changing the circuit, without changing its functionality.

There are two ways to make a circuit better testable:

1. Redesigning the circuit (Subsection 2.8.1)
2. Inserting test points (Subsection 2.8.2)

2.8.1 Circuit mutation: Redesigning the circuit

In case the CUT is very poorly PR testable, and all state-of-the-art BIST techniques cannot improve the PR testability without too much hardware overhead, one can consider to redesign the CUT such that it becomes better PR testable. Redesigning the circuit should take place without changing its functionality. Redesigning the circuit impacts the time-to-market of a CUT, because a large part of the design flow has to be done again, possibly in multiple iterations; e.g., verification of functionality, synthesis, timing analysis, ATPG or BIST implementation, etc.. Therefore redesigning the circuit can be seen as a very last solution in order to improve the testability of a circuit. The necessity of redesigning a circuit can easily be avoided when circuit designers take into account Design-for-Test rules [Fuj85], like the BIST restrictions described in Section 2.3.

2.8.2 Circuit mutation: Inserting test points

Another way to make a circuit better testable is by inserting TPs. Instead of redesigning the complete circuit, extra inputs (CPs) and/or outputs (OPs) are added to make internal parts of the circuit better testable, without changing the remaining circuit structure. Also after TPI, parts of the design flow has to be redone; i.e., the timing analysis (to make sure that the circuit achieves its required performance), synthesis, ATPG and/or BIST implementation have to be redone, as is described in [Het99, Fei99, Fei01]. However the impact on the design flow and time-to-market is limited compared to redesigning the complete circuit. As will be shown in the following chapters, TPI results in limited hardware overhead while offering significant fault coverage improvements for BIST.

2.9 Summary and conclusions

The purpose of BIST is to embed a test of a circuit on-chip in order to reduce external ATE requirements. However, due to RPR faults in the circuits, the fault coverages achievable with BIST are often not high enough to meet the quality requirements of the semiconductor industry. Also a successful BIST implementation implies several restrictions on a circuit, especially with respect to three-state elements.

In order to improve BIST fault coverages, one can modify the test applied and/or modify the circuit such that it becomes better testable. Several state-of-the-art BIST methods have been described with which higher fault coverages can be achieved. These state-of-the-art BIST methods have been divided into methods that modify the test patterns generated by a PRTPG, methods that embed a full set of deterministic test patterns in the TPG and methods that embed parts of deterministic test patterns in the TPG.

The BIST methods that modify the generated test patterns are capable of increasing the fault coverage. However, especially for RPR circuits, reaching higher or even complete fault coverage results in huge hardware overhead. The BIST methods that embed a full set of deterministic test patterns are capable of reaching complete fault coverage. Their hardware overhead depends on the size of the deterministic test set they embed. Only for circuits with a very small deterministic test set, these methods can implement BIST with affordable hardware overhead. Results on ISCAS circuits have shown that this overhead can become larger than the circuit itself when the deterministic test set is not small. The BIST methods that embed parts of deterministic test patterns can also increase the fault coverages up to complete fault coverage. Like with the methods that modify the test patterns, their hardware overhead increases for poorer PR-testable circuits.

When the hardware-overhead of the state-of-the-art BIST methods is too high for successful BIST implementation, one can consider redesigning the circuit to make it better testable. A redesign of the circuit has significant impact on the design-process and results in a delayed time-to-market. Instead of a complete redesign of the circuit, also TPs can be inserted to improve the testability of internal parts of the circuit. In the next two chapters it will be shown that with TPI, also high PR fault coverages can be achieved with only limited hardware overhead.

CHAPTER 3

Test Point Insertion

This chapter starts with a general description of *Test Point Insertion (TPI)*. What is TPI, what are its advantages and what are its limitations? Section 3.1 introduces test points (TPs) and Section 3.2 describes how TPI can facilitate testing. Section 3.3 describes in general how TPI methods select the positions in the circuit where TPs should be inserted. Section 3.4 describes the requirements and limitations of TPI with respect to industrial designs. The testability analysis method COP (Controllability/Observability Program) [Brg84] is often used in TPI algorithms for determining the PR testability of faults in a circuit. Also most of the TPI algorithms for improving BIST fault coverage are based on a STUMPS (Self-Test Using a MISR/parallel SRSG) [Bar87] architecture. Therefore, before several state-of-the-art TPI algorithms, which mainly focus on improving BIST fault coverage, are described in Section 3.7, first descriptions of COP and STUMPS are given in Section 3.5, respectively Section 3.6. In the following chapters, new TPI techniques and algorithms will be proposed. An overview of the TPI topics addressed in the following chapters, including a description of the TPI benchmark circuits that have been used to evaluate the proposed techniques and algorithms, is given in Section 3.8. Section 3.9 concludes this chapter.

3.1 Test points

The key idea of TPI is to insert extra logic into the circuit such that internal signal lines in the circuit will become better controllable and/or observable. This extra logic will result in better testability of internal parts of the circuit, however it should not influence the functionality of the circuit in normal operation mode.

Traditionally, *test points (TPs)* can be categorized into observation points and control points. An *observation point (OP)* is an additional primary output which is inserted into the circuit in order to obtain a better *observability* for some signal lines in the circuit. The *observability* for a signal line l is the ‘*difficulty*’ to propagate a value change at line l to a PO. The exact definition of ’difficulty’ depends on the used testability analysis method.

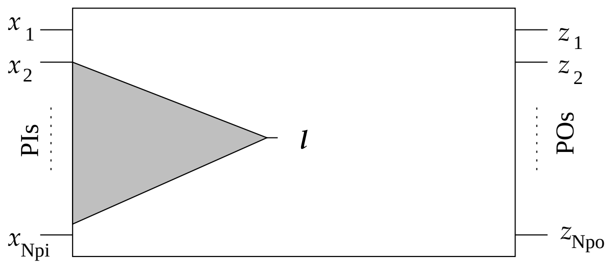


Figure 3.1: Region of influence of an observation point at line l

The impact of an OP inserted at an internal line l of a given circuit is shown in Fig. 3.1; it improves the observability of line l itself and the observabilities of lines in the fan-in cone of line l. Improves means that it becomes easier, or less difficult, to propagate a value change to a PO.

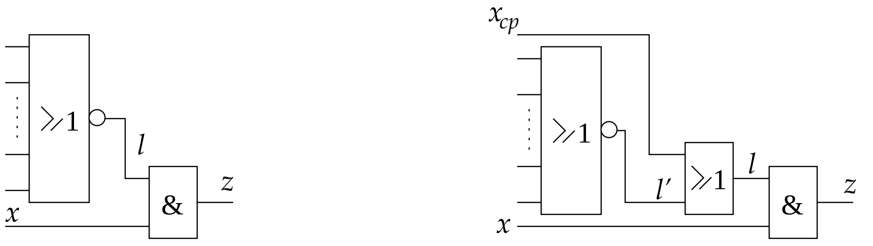
A *control point (CP)* is an additional primary input plus additional logic inserted in the CUT in order to obtain a better *controllability* for parts of the circuit. The *controllability* of a signal line l in the circuit is the '*difficulty*' to set line l to value v. Controllability can be split into *1-controllability* and *0-observability*. The *1(0)-controllability* of a line l is the '*difficulty*' to get a '1(0)' on l.

Unlike an OP, which only affects the observability of a the fan-in cone of the OP, a CP influences both the controllability and the observability of regions in the CUT. The observability of a signal line depends on the controllability of other lines. Hence a controllability change will also result in observabilities changes. This is illustrated in Fig. 3.2. A 0/1 discrepancy at signal line x in Fig. 3.2(a) is only observable on output z when l is 1. When l is 0, the value on z is 0, no matter the value of x. By changing the controllability of l by the insertion of a CP at l, as illustrated in Fig. 3.2(b), it becomes easier to get a 1 on l (just put a 1 on input x_{cp}) and hence it becomes also easier to be able to observe a 0/1 discrepancy at signal line x on z; the observability of x changes due to the controllability change of l.

However, a CP also results in reduced observability in the fan-in cone of the inserted CP. Compared to line l in Fig. 3.2(a), it has become more difficult to observe a 0/1 discrepancy at line l' on an output. In addition to the requirements without the CP, a 0 is required on input x_{cp} in order to propagate a 0/1 discrepancy to an output.

Fig. 3.3 shows the impact of a CP on a circuit. The hatched area (I) of Fig. 3.3 indicates the region of the circuit where the controllabilities of lines change because of the controllability improvement of line l. The shaded area (II) sketches the region of the circuit where observabilities of lines alter as a consequence of the changes in the controllabilities.

In the IC industry, AND/OR-gate CPs and OPs are controlled and observed by extra SFFs in the scan-chain. But when these SFFs are necessary anyway, can they be used as



(a) A part of a circuit (b) OR gate as CP inserted at signal line *l*

Figure 3.2: Impact of a control point on the observability

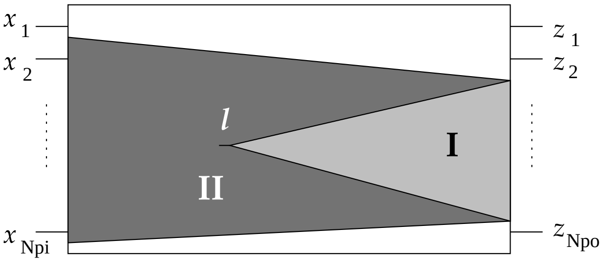


Figure 3.3: Regions of influence of a control point at line *l*

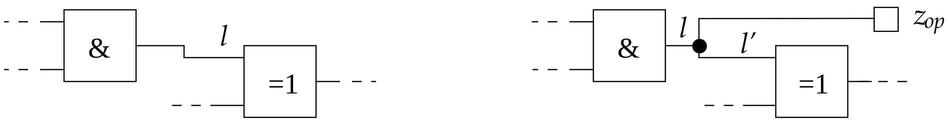
CPs and OPs themselves such that the AND/OR-gates are not necessary any longer? This is indeed possible, as long as *transparent SFFs (TSFFS)* are used, which act like buffers during normal operation and as SFF during test mode.

Fig. 3.4 shows an example of the implementation of a transparent SFF compared to the traditional AND/OR TPs. Fig. 3.4(a) shows the circuit part without TPs, Fig. 3.4(b) shows the circuit after the insertion of an OP at *l* which can be observed on output z_{op}. Fig. 3.4(c) shows the circuit after the insertion of a CP at line *l* which can be controlled with input x_{cp}. Fig. 3.4(d) shows the circuit (in test mode) after the insertion of a transparent SFF. The output of signal line *l* is connected with the SFF input such that *l* becomes fully observable, increasing the observabilities in the input-cone of *l*. The output of the SFF is connected with signal line *l'*. Signal line *l'* becomes controllable, like all PIs.

TSFFs have two properties different from AND/OR TPs:

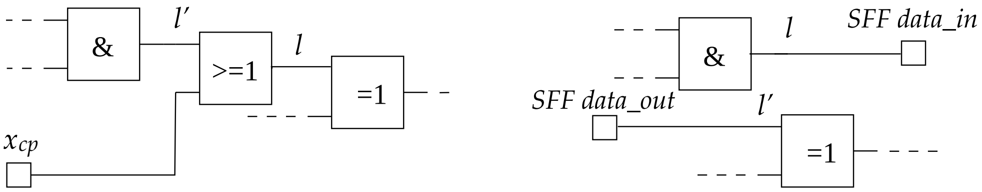
1. The inserted TSFF will act as a CP *and* an OP at the same time; you cannot insert a CP apart from an OP. The TSFF will control the output cone of the line where it is inserted and will observe the input cone of this line.

2. TSFFs will not (lead to a) decrease (of) the observability in their fan-in cones, which is the case for AND- and OR-gate CPs. To the contrary, because a SFF TP



(a) Circuit before TPI

(b) OP at line *l*



(c) OR-gate as CP at line *l*

(d) Transparent SFF at line *l*

Figure 3.4: Traditional TPs compared to the transparent SFF

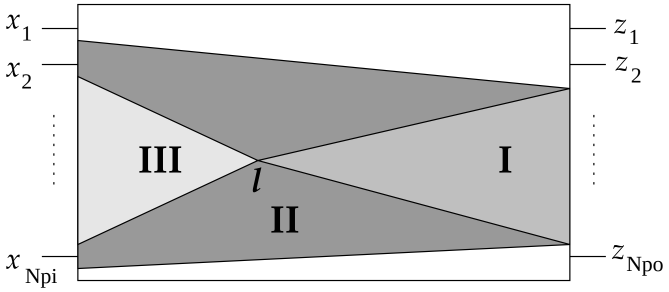


Figure 3.5: Regions of influence of a TSFF at line *l*

can observe its input, the line connected to the SFF will become fully observable.

Fig. 3.5 shows the impact of a TSFF on a circuit. Area I of Fig. 3.5 indicates the region of the circuit where the controllabilities of lines change because of the improvement in controllability at *l* due to the SFF. Area II sketches the region of the circuit where lines observabilities alter as a consequence of the changes in the controllabilities. Area III indicates the region where the observability in the circuit changes due to the observability improvement at *l* and (possibly) also due to the controllability changes.

3.2 TPI to facilitate testing

As already mentioned in Section 1.3, TPI can be used to facilitate testing by solving testability problems within a design. The main objective of TPI algorithms is to determine which lines in a circuit are the most efficient places for inserting a TP, in order to improve the testability and to minimize the extra silicon overhead caused by the inserted TPs. TPI can facilitate testing in different ways. TPI can be used to:

1. facilitate pseudo-random testing:
   In case of on-chip testing/BIST, TPI can improve the PR testability of the RPR faults in a circuit such that higher PR fault coverages can be achieved with fewer PR patterns. TPI methods targeted at facilitating PR testing, i.e., for BIST, are given in Chapter 4.

2. facilitate ATPG
   In case of using ATPG generated patterns, for off-chip testing or in BIST with an embedded deterministic test set, TPI can facilitate the generation of ATPG test patterns for the faults within a circuit. TPI methods to facilitate ATPG are given in Chapter 5.

TPI is not limited only to facilitate testing of stuck-at faults. In Chapter 6 it will be shown that TPI also facilitates the generation of patterns for other fault models, e.g., for gate-delay faults.

3.3 Test point selection

TPI algorithms insert TPs to improve the testability of certain (groups of) faults in the circuit. As described in [Geu97b], the TPI algorithms can be divided into three categories, based on the method used to determine the faults which testabilities should be improved:

1. Methods that are based on **ATPG**
2. Methods that are based on **fault simulation**
3. Methods that are based on **testability analysis**

The ATPG based TPI algorithms use ATPG results to determine the faults for which the testability should be improved. These are the faults for which the ATPG tool had trouble generating a test pattern or could not even generate a test pattern. The TPI methods based on fault simulation use the information of fault simulation on a set of (PR) test patterns to select the set of faults of which testability should be improved.

The testability measures, e.g., COP [Brg84], SCOAP [Gol80] or test counts [Hay74], found by testability analysis, reflect the testability of a given circuit and its faults. The TPI methods based on testability analysis use this information to select the set of faults for which testabilities should be improved. Determining the testability measures is less CPU-time consuming than fault simulation, but they are less accurate than the fault simulation results.

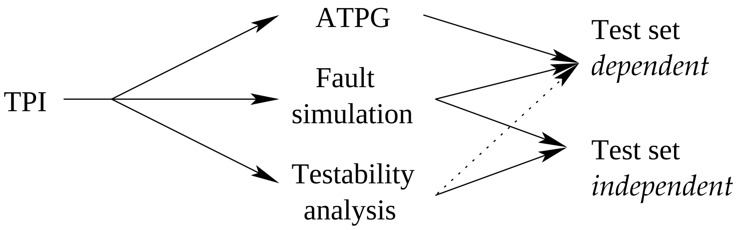


Figure 3.6: The different possible ways in doing TPI

Two different methods exist to determine the most convenient places to insert a TP. *Test set independent* TPI methods only use the information of the CUT, no information is used about the test set which is applied to the CUT. These methods assume that all inputs of the CUT have an equal probability for a 0 and a 1. With these probabilities, the detectability estimates of all faults are computed.

However *test set dependent* TPI methods do use the information of the test set. The test set can provide the probability for a 0 or a 1. This results in different but more accurate detectability estimates. Also fault simulation of the test patterns in the set can be used to determine the undetected or hard-to-detect faults for which a TP should be inserted.

Fig. 3.6 shows the different ways TPI can be implemented. Each path represents a method. The TPI methods based on ATPG use the ATPG test set information in the TP selection; they are always test set dependent, hence no arrow from *ATPG* to *Test set independent*. The methods based on fault simulation can be both test set dependent and independent; they depend on a fault simulation run of a pre-defined test set or on a PR test run. In general, the methods based on testability analysis are not based on a test set and so are test set independent. However, this is not necessarily true, using the test set information for the probabilities for a 1 at the inputs of the circuit to compute the detectabilities and testability measures, is an example of a test set dependent method based on testability analysis, therefore the dotted arrow from *Testability analysis* to *Test set dependent*.

Test set independent TPI methods can be divided further into methods that use:

1. Global optimization
   These methods use measures (costs) that reflect the total testability within the circuit. The TP that is selected to be inserted is the TP that results in the best testability improvement (cost reduction) in the whole circuit. Although, according to the cost measures, the selected TP will globally result in the best testability improvement, it might not solve specific local testability problems.

2. Local optimization
   Local optimization solutions only try to solve specific local problems in the circuit, without taking into account the global impact of the TP.

3.4 TPI for industrial circuits

There exist many TPI algorithms [Sav91, Sei91, Che95b, Sch95, Tam96, Tou96, Tsa97], as will be described in the following sections. However, not all of these algorithms are capable to cope with industrial circuits. E.g., Seiss [Sei91] uses equations to calculate the best TP positions, based on the assumption that lines that are not 1, are 0. Because lines in industrial circuits can also become Z or U, the outcome of the TP selection can result in selected TPs that do not result in better testability of the circuit. TPI can only be used in the semiconductor industry when the TPI methods can be used in combination with industrial circuits.

Another problem with TPI for industrial circuits are possible bus-conflicts after TPI. Circuits containing buses can be designed in such a way that bus-conflicts will never occur, to avoid circuit damage and/or to implement the circuit with BIST. When TPs are inserted at wrong chosen positions, it is possible that after the insertion buses can become in conflict, resulting in circuit damage or the inability of proper BIST insertion.

Industrial circuits also often contain embedded memories. During test, the values at the inputs of the circuit from these embedded memories are often unknown. These unknown values propagate through the circuit resulting in poorer testability of the circuits. Test circuitry or TPs can be inserted such that these unknown inputs are bypassed and can be controlled by the test circuitry resulting in an improvement of the testability of the circuit.

TPs cannot be inserted at every location within an industrial design. Due to the insertion of a TP, the critical path[1] of a design might increase. As a result, the IC has to operate on a lower clock frequency and might not reach the performance it was originally designed for. TPI methods for industrial circuits should be able to exclude lines to be TP candidate, e.g., to exclude positions that increase the critical path length of the circuit.

3.5 Controllability/Observability Program (COP)

This section describes the *controllability / observability program (COP)* [Brg84] that is often used in TPI algorithms [Sav91, Sei91, Tsa97] to determine the TP positions. Section 3.5.1 introduces the COP controllability and observability values. In Section 3.5.2 the equations for the controllability and observability for the in- and outputs of the standard gates are given.

3.5.1 Introduction to COP

COP can be used to estimate the probability that a SAF will be covered by a PR pattern and hence determine which faults are RPR. This testability information can be used

---

[1] the path in the circuit that determines the maximum clock-frequency (performance), i.e., the path with the longest delay

by a TPI algorithm to determine where in the circuit TPs should be inserted, such that the number of RPR faults decreases and the overall testability of the circuit improves. COP assumes that the circuit conforms to a full-scan design.

For each signal line l in the circuit, COP provides statistical values for the controllability C_l and observability W_l of that line. In COP, the *controllability* (or the *1-controllability ($C1_l$)*) of a signal line l is defined as the *probability* that line l is 1. The *0-controllability ($C0_l$)* of a signal line l is the *probability* that line l is 0. In case of a Boolean circuit, if a line is not one, it is zero:

$$\begin{aligned} C0_l + C1_l &= 1 \text{ (Boolean circuit)} \\ C0_l &= 1 - C1_l \end{aligned} \tag{3.1}$$

Given a large number of PR patterns, each PI has an equal probability for a 1 and for a 0, hence the controllabilities of the PIs of the circuit are initialized at 0.5: C_{PI} = 0.5. The controllabilities of the other signal lines are circuit dependent and their calculation is described in Subsection 3.5.2.

The *observability* of a signal line l is defined as the *probability* that a value change on line l will lead to a value change on at least one PO. The observability of all POs is 1: $W_{PO} = 1$; the observabilities of the other lines depend on the structure of the circuit and their calculation is given in Subsection 3.5.2.

Compared to the description of controllability and observability as mentioned in Chapter 3, COP defines the *difficulty* to control a line as the *probability* the line is 1(0) and defines the *difficulty* to observe a line on a PO as the *probability* that a value discrepancy on that line is visible on a PO.

A very large controllability of a line l, i.e., very close to 1, indicates that it is hard to test the SA1 fault on l. In that case the probability that l is 0 (required to test for the SA1 fault), is very small. The opposite is true when the controllability of l is very close to 0 (the 0-controllability is very close to 1). In this case it is hard to test the SA0 fault. When the observability of a line l is close to 0, this means that it is very hard to propagate a value discrepancy of line l to a PO.

A TP can be inserted to improve the 1-controllability, 0-controllability and/or observability of these lines.

3.5.2 Calculation of the COP controllability and observability

Fig. 3.7 shows a *2-input AND*-gate and a *2-input OR*-gate. Output z of the AND-gate is only 1, when both x and y are 1. Hence, The COP controllability of the 2-input AND-gate becomes:

$$C_z = C_x \cdot C_y \tag{3.2}$$

Output z of the OR-gate is 1, when at least one of the inputs x and y is 1. Output z is 0, when both x and y are 0. Hence the COP 0-controllability of the 2-input OR-gate

Figure 3.7: An AND-gate and an OR-gate.

becomes:

$$C0_z = C0_y \cdot C0_y$$

Rewriting in COP 1-controllabilities (assuming Boolean circuit) this results in:

$$C_z = 1 - (1 - C_x) \cdot (1 - C_y) \tag{3.3}$$

A discrepancy on an input x of a gate can only be observed when the other inputs of the gate have a *non-controlling value*. A *controlling value* of a gate is a value which forces the output of the gate to a specific value, in spite of the values of all other inputs. In the case of an AND-gate, the controlling value is 0, in case of an OR-gate, the controlling value is 1.

A value discrepancy on input x of the AND-gate in Fig. 3.7 can only be observed when input y has the non-controlling value 1 and output z is observable. Hence the observability of x becomes:

$$W_x = C_y \cdot W_z \tag{3.4}$$

Input x of the OR-gate of Fig. 3.7 can only be observed when input y has the non-controlling value 0 and output z is observable. Hence the observability of input x becomes:

$$\begin{aligned} W_x &= C0_y \cdot W_z \\ &= (1 - C_y) \cdot W_z \text{ (Boolean circuit)} \end{aligned} \tag{3.5}$$

These equations for COP controllability and observability can be extended for gates with more than 2 inputs; for an L-input AND-gate the COP controllability and observability become:

$$\begin{aligned} C_z &= C_{x_1} \cdot C_{x_2} \cdot \ldots \cdot C_{x_L} \\ W_{x_1} &= C_{x_2} \cdot C_{x_3} \cdot \ldots \cdot C_{x_L} \cdot W_z \end{aligned}$$

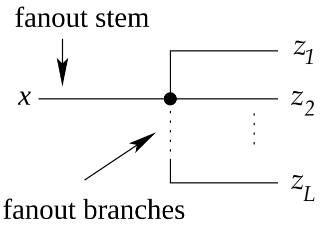
where x_1, x_2, …, x_L are the inputs of the L-input gate and z is the output of the gate.

For an L-input OR-gate the controllability and observability become:

$$\begin{aligned} C0_z &= C0_{x_1} \cdot C0_{x_2} \cdots C0_{x_L} \\ C_z &= 1 - (1 - C_{x_1}) \cdot (1 - C_{x_2}) \cdots (1 - C_{x_L}) \text{ (Boolean circuit)} \\ W_{x_1} &= C0_{x_2} \cdot C0_{x_3} \cdots C0_{x_L} \cdot W_z \\ W_{x_1} &= (1 - C_{x_2}) \cdot (1 - C_{x_3}) \cdots (1 - C_{x_L}) \cdot W_z \text{ (Boolean circuit)} \end{aligned}$$

Table 3.1: COP testability measures for standard gate types.

Gate	Controllability C_z	Observability W_{x_1}
AND	$C_{x_1} \cdot C_{x_2}$	$C_{x_2} \cdot W_z$
NAND	$1 - C_{x_1} \cdot C_{x_2}$	$C_{x_2} \cdot W_z$
OR	$1-(1-C_{x_1})(1-C_{x_2})$	$(1-C_{x_2}) \cdot W_z$
NOR	$(1-C_{x_1})(1-C_{x_2})$	$(1-C_{x_2}) \cdot W_z$
XOR	$C_{x_1} + C_{x_2} - 2 \cdot C_{x_1} \cdot C_{x_2}$	W_z
NXOR	$1 - C_{x_1} - C_{x_2} + 2 \cdot C_{x_1} \cdot C_{x_2}$	W_z
BUF	C_{x_1}	W_z
INV	$1 - C_{x_1}$	W_z



$$C_{z_1} = C_{z_2} = \cdots = C_{z_L} = C_x \tag{3.6}$$

$$W_x = 1 - (1 - W_{z_1}) \cdot (1 - W_{z_2}) \cdots (1 - W_{z_L}) \tag{3.7}$$

Figure 3.8: The COP measures for a fan-out stem

Table 3.1 shows the COP testability measures for standard gate elements. x_1 and x_2 are the gate inputs and z is the gate output. The equations in Table 3.1 are only exact when the values on the gate inputs are independent. This is not the case when a line fans out and reconverges further on in the circuit. This will lead to a small error in the calculations of the COP controllabilities and observabilities.

But what does happen on a fan-out node? From the signal line, there exist multiple paths to the POs, see Fig. 3.8. The controllabilities of the fan-out branches are equal to the controllability of the fan-out stem. A fault on x can be observed when the fault can be observed on at least one of the fan-out branches $z_1 \cdots z_L$ and thus is only not observable when none of the fan-outs can be observed. The observability of the lines on the fan-out stem is the OR function of the observabilities of all fan-out branches.

The COP controllabilities and observabilities in the circuit are calculated as follows:

1. Initialize the COP controllabilities of all PIs (and SFF-outputs) to 0.5.

2. Propagate the controllabilities from the PIs forward to the POs using the equations given in the Column *Controllability* C_z in Table 3.1.

3. Initialize the COP observabilities of all PIs (and SFF-inputs) to 1.

4. Propagate the observabilities from the POs backward to the PIs using the equations given in the Column *Observability* W_x in Table 3.1.

The COP controllabilities and observabilities can be combined to find the *COP detection probability* for a given fault. The COP detection probability of SAF *f* on a line *l* is the probability that *f* will be detected on a PO. A SA1 fault on a line *l* can only be detected when *l* is 0 in the fault-free circuit *and l* is observable. The detection probability estimate of a SA1 fault can be calculated with Eq. 3.8.

$$\begin{aligned} Pd_{l/SA1} &= C0_l \cdot W_l \qquad (3.8) \\ &= (1-C_l)\cdot W_l \text{ (Boolean circuit)} \end{aligned}$$

A SA0 fault on line *l* can only be detected when *l* is 1 in the fault-free circuit and line *l* is observable on a PO. The detection probability estimate of a SA0 fault can be calculated with Eq. 3.9.

$$Pd_{l/SA0} = C_l \cdot W_l \qquad (3.9)$$

Obviously, the larger the detection probability of a fault, the easier it is to detect this fault with a PR pattern. The difficult faults for PR testing are the faults with a very low detection probability. The faults with a large detection probability will probably be detected with only a few PR test patterns, while the RPR faults, with a low detection probability, require a very large number of PR patterns in order to have a reasonable probability that the fault will be covered by a test pattern.

3.5.3 Cost function and the cost gradient values

The goal of TPI for BIST is to obtain a maximal improvement in the PR testability of a circuit with as few TPs as possible. During the selection of the TP positions, it is not advisable to exclusively rely on the COP controllabilities and observabilities, since due to their local nature, they are lacking the capability to analyze and describe the circuit's testability problems from a more global point of view. For example given that line *l* in Fig. 3.3 has a small controllability C_l. Increasing C_l will not necessarily lead to an improvement of the circuit's overall testability. This change may deteriorate the detection probabilities of several faults in the shaded area of Fig. 3.3 and may thus have a negative impact on the RPR testability of the circuit.

In [Lis87], a cost function (CF) is introduced, which represents the global testability of the circuit and the testability of individual faults. It is defined as the summation of the inverses of all COP detection probabilities. When a fault is easy to detect, its detection probability is large and the inverse is small, which means that the impact of this fault on the cost is minimal. On the contrary, a fault with a very small detection probability has a large inverse and thus a high impact on the cost.

The equation for the CF given in [Lis87] is:

$$K = \frac{1}{F} \cdot \sum_{f=1}^{F} K_f \tag{3.10}$$

$$K_f = \frac{1}{Pd_f} \tag{3.11}$$

where K is the global cost function of the CUT, F is the number of stuck-at faults, K_f is the cost contribution of fault f, and Pd_f is the detection probability of a fault f in the CUT. A large circuit, with many faults, is not necessarily harder to test than a small circuit, with fewer faults. To compensate the CF for the number of faults in the CUT, the CF is multiplied with $\frac{1}{F}$. However, many TPI algorithms that use this CF [Sei91, Tsa97] ignore the $\frac{1}{F}$ factor and use Eq. 3.12.

$$K = \sum_{f=1}^{F} K_f \tag{3.12}$$

For every signal line l in the CUT, two SAFs exist, e.g., the SA0 and SA1 fault. Therefore Eq. 3.10 can also be written as Eq. 3.13.

$$\begin{aligned} K &= \frac{1}{2L} \cdot \sum_{l=1}^{L} \left(K_{l/SA0} + K_{l/SA1}\right) \\ &= \frac{1}{2L} \cdot \sum_{l=1}^{L} \left(\frac{1}{Pd_{l/SA0}} + \frac{1}{Pd_{l/SA1}}\right) \end{aligned} \tag{3.13}$$

where L is the total number of signal lines in the CUT and *l/SA0* and *l/SA1* represent line l SA0, respectively SA1. Ignoring the $\frac{1}{2L}$ factor, Eq. 3.13 becomes:

$$\begin{aligned} K &= \sum_{l=1}^{L} \left(K_{l/SA0} + K_{l/SA1}\right) \\ &= \sum_{l=1}^{L} \left(\frac{1}{Pd_{l/SA0}} + \frac{1}{Pd_{l/SA1}}\right) \end{aligned} \tag{3.14}$$

[Lis87] also introduces the cost gradient values. They are used to estimate the impact of controllability and observability changes of a line line on the rest of the circuit. For each line, two *gradient values* are defined, namely $\frac{dK}{dC_l}$ and $\frac{dK}{dW_l}$. They are the derivatives of the CF K with respect to the 1-controllability, respectively the observability, of line l

and are called the *controllability gradient* and the *observability gradient* of line l.

Both gradient values are still not very good indicators of the global testability impact after the insertion of a TP, because they represent the change rate of K to C_l and W_l respectively due to an infinitely small change of C_l and W_l, whereas the insertion of a TP may cause C_l or W_l to change drastically. Although the gradients are not very accurate themselves, they still can be useful in determining good TP positions, as will be shown in Section 3.7. How to calculate the gradient values is described in the following text.

Calculation of the gradient values

Changing a line's observability only affects the detection probabilities of its predecessors, see Fig. 3.1. Therefore a change in the observability of PI x affects only the detection probability of the input itself and so only two terms of the global cost function; the term with respect to the SA0 fault, respectively the SA1 fault, on input x. The derivative of the CF with respect to a change in *PI observability* of input x becomes:

$$\frac{dK}{dW_x} = \frac{dK_{x/SA0}}{dW_x} + \frac{dK_{x/SA1}}{dW_x} \tag{3.15}$$

In case Eq. 3.11 is used for the cost contribution for a fault, Eq. 3.15 becomes:

$$\begin{aligned} \frac{dK}{dW_x} &= \frac{d(\frac{1}{Pd_{x/SA0}})}{dW_x} + \frac{d(\frac{1}{Pd_{x/SA1}})}{dW_x} = \frac{d(\frac{1}{C_x \cdot W_x})}{dW_x} + \frac{d(\frac{1}{(1-C_x)\cdot W_x})}{dW_x} \\ &= \frac{-C_x}{(C_x \cdot W_x)^2} + \frac{C_x - 1}{((1-C_x)\cdot W_x)^2} = \frac{-C_x}{Pd^2_{x/SA0}} + \frac{C_x - 1}{Pd^2_{x/SA1}} \end{aligned} \tag{3.16}$$

Given a gate with X inputs and one output, changing the observability on the gate output z also results in observability changes on the gate inputs, because the observabilities on the gate inputs depend on the observability at the gate output (see Column *Observability* W_x in Table 3.1). Thus the CF will not only change by the two terms with respect to the SA0 and SA1 fault of output z, but also by the observability changes on the X gate inputs. Since it is known how the cost changes with respect to the observability on these gate inputs, it is possible to apply a chain-rule to compute the derivative with respect to observability changes on the gate output z. The derivative of the CF with respect to observability changes on the gate output z is given in Eq. 3.17, where x_i represents the ith input of the gate.

$$\frac{dK}{dW_z} = \frac{dK_{z/SA0}}{dW_z} + \frac{dK_{z/SA1}}{dW_z} + \sum_{i=1}^{X} \frac{dK}{dW_{x_i}} \frac{dW_{x_i}}{dW_z} \tag{3.17}$$

In case Eq. 3.11 is used as cost contribution for a fault, Eq. 3.17 can be written as:

$$\frac{dK}{dW_z} = \frac{-C_z}{Pd^2_{z/SA0}} + \frac{(C_z - 1)}{Pd^2_{z/SA1}} + \sum_{i=1}^{X} \frac{dK}{dW_{x_i}} \frac{dW_{x_i}}{dW_z} \tag{3.18}$$

The first two terms of Eq. 3.17 are due to the observability changes on output z itself, and the summation is due to the observability changes on the X inputs of the gate. The quantities dW_{x_i}/dW_z depend on the gate type and represent a transfer function that dictates how a gate's input observability will change, given a change in output observability. They can be calculated from the gate observability equations given in Table 3.1.

After the computation of dK/dW for all signal lines in the CUT, the dK/dC values, which depend on dK/dW, can be determined. Because changing a gate's input controllability affects both the output controllability and the observability of the other gate inputs (see Fig. 3.3), the chain-rule equation for dK/dC is slightly more complicated than for the dK/dW forward propagation. However, changing the controllability on a PO only influences the cost contribution for the SA0 and SA1 faults on that PO. The derivative of the CF K with respect to a controllability change on PO z is:

$$\frac{dK}{dC_z} = \frac{dK_{z/SA0}}{dC_z} + \frac{dK_{z/SA1}}{dC_z} \tag{3.19}$$

And using Eq. 3.11 as cost contribution for a fault, Eq. 3.19 becomes:

$$\frac{dK}{dC_z} = \frac{d(\frac{1}{Pd_{z/SA0}})}{dC_z} + \frac{d(\frac{1}{Pd_{z/SA1}})}{dC_z} = \frac{-W_z}{Pd^2_{z/SA0}} + \frac{W_z}{Pd^2_{z/SA1}} \tag{3.20}$$

The two terms in Eq. 3.20 represent the cost change caused by the the SA0 and SA1 detectabilities changes on PO z. Propagation of dK/dC proceeds in backward direction, beginning with initialized POs, according to the following equation:

$$\frac{dK}{dC_{x_j}} = \frac{dK_{x_j/SA0}}{dC_{x_j}} + \frac{dK_{x_j/SA1}}{dC_{x_j}} + \sum_{i=1, i\neq j}^{X} \frac{dK}{dW_{x_i}} \frac{dW_{x_i}}{dC_{x_j}} + \sum_{k=1}^{Z} \frac{dK}{dC_{z_k}} \frac{dC_{z_k}}{dC_{x_j}} \tag{3.21}$$

where X are the number of inputs of the gate, Z the number of outputs, x_i and x_j are the gate inputs and z_k are the gate outputs. Using Eq. 3.11 as cost contribution for a fault, Eq. 3.21 becomes:

$$\frac{dK}{dC_{x_j}} = \frac{-W_{x_j}}{Pd^2_{x_j/SA0}} + \frac{W_{x_j}}{Pd^2_{x_j/SA1}} + \sum_{i=1, i\neq j}^{X} \frac{dK}{dW_{x_i}} \frac{dW_{x_i}}{dC_{x_j}} + \sum_{k=1}^{Z} \frac{dK}{dC_{z_k}} \frac{dC_{z_k}}{dC_{x_j}} \tag{3.22}$$

The first two terms calculate the cost impact of controllability changes at line x_j itself. The third term, the summation over all other inputs of the gate, gives the cost impact of the observability changes on these other inputs due to the controllability change on input x_j. The last term, the summation over all outputs, gives the cost impact of the controllability changes on the gate outputs due to the controllability changes on input x_j. The dW_{x_i}/dC_{x_j} and dC_{z_k}/dC_{x_j} values, which represent the observability change at input x_i, respectively controllability change on output z_k given the controllability change on x_j, can

Algorithm 3.1 COP gradients calculation

```
Input:  CUT, C and W for all lines in CUT
Output: COP gradients for all lines in CUT
for all PIs $\in$ CUT do
    Initialize $dK/dW$ with Eq. 3.16.
for all Lines $\in$ CUT from PIs to POs do
    Compute $dK/dW$ for all other lines by applying Eq. 3.18 to all gates in the circuit a single forward pass.
for all POs $\in$ CUT do
    Initialize $dK/dC$ with Eq. 3.20.
for all Lines $\in$ CUT from POs down to PIs do
    Compute $dK/dC$ for all other lines by applying Eq. 3.22 to all gates in a single backward pass.
```

be derived from the equations listed in Table 3.1.

The complete gradients calculation procedure is given in Algorithm 3.1. This procedure is linear in the number of gates in the CUT, since the equations are applied on a gate-by-gate basis.

3.6 STUMPS

In this dissertation, it is assumed that the circuit uses the popular *STUMPS* approach in case of PR BIST. *STUMPS* is an abbreviation for *Self-Test Using MISR / Parallel SRSG*; where *SRSG* stands for *Shift Register Sequence Generator*, see [Bar87]. Originally, STUMPS was a BIST architecture for an *LSSD (Level Sensitive Scan Design)*-based multi-chip system, in which each *Field Replaceable Unit (FRU)* contains a large number of logic chips. But the STUMPS model can also be used in a single chip design, it is just a different scale; the circuit represents the FRU and the different scan-chains in the circuit represent the large number of logic chips. This section describes the properties of STUMPS and how it can be adapted for PR BIST.

3.6.1 Introduction to STUMPS

The STUMPS architecture often used in PR BIST is depicted in Fig. 3.9. STUMPS BIST is a full-scan BIST method. It is assumed that the FFs of the circuit core are connected into multiple scan-chains (*scan-chain*). The PRTPG (implemented by means of an LFSR or CA) applies PR test patterns to the core by shifting the scan-chains. While the new patterns are shifted into the chains by the LFSR/CA, the output of the scan-chains

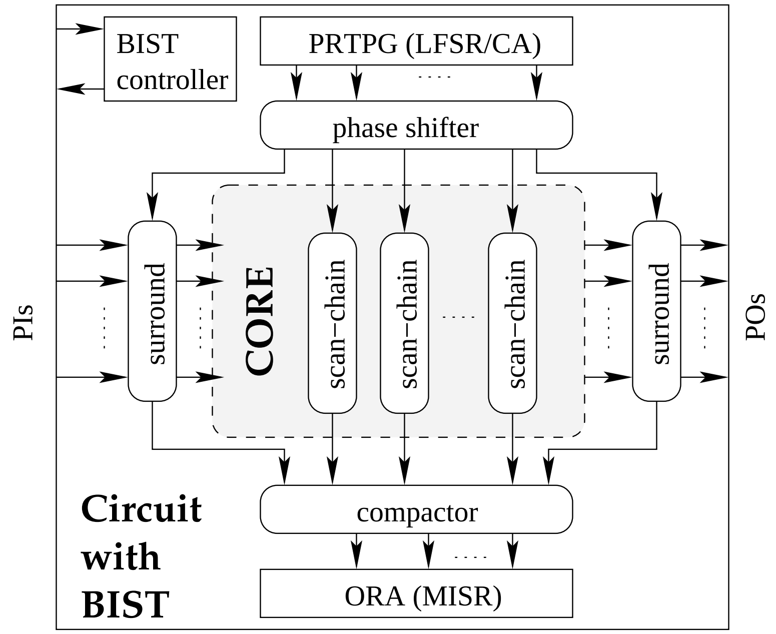


Figure 3.9: The STUMPS architecture for PR BIST

are captured by the MISR. After a pattern has been shifted in, the CUT is clocked once such that all FFs contain the new state of the CUT. The state caught in the chains is shifted out into the MISR and meanwhile the new input pattern is shifted in. Often the PIs and POs are also connected into a surrounding scan-chain (*surround*), such that they can be controlled and be observed by the LFSR, respectively MISR, just like the FFs.

After all patterns have been applied, the state of the MISR is compared to the state of a MISR of the fault-free circuit, and the BIST controller returns a signal to inform whether the core is fault-free.

In Fig. 3.9 also a *phase-shifter* and a *compactor* are shown. A phase-shifter can be added to reduce the correlation between the bits shifted into the scan-chains, see Subsection 3.6.2. The compactor can be added to reduce the number of bits shifted into the MISR and hence reduce the size of the MISR.

In Fig. 3.9 it is assumed that the PIs and POs are connected in surrounding scan-chains or boundary scan-chains such that they can be controlled or observed in a similar way as the internal scan-chains. It is also possible that the PIs are controlled by a separate LFSR and the PO results are captured into a separate MISR.

3.6.2 Restrictions and problems of STUMPS BIST

The advantage of STUMPS BIST, using an LFSR/CA to apply test patterns and using a MISR to capture the responses, is the low hardware overhead. But STUMPS BIST also introduces several problems and restrictions for the CUT. In the following text the problems for STUMPS BIST are briefly discussed and possible solutions are mentioned.

The problems in a STUMPS BIST environment are:

1. *Lower fault coverage caused by RPR faults*.
   LFSRs and CAs often fail to detect the RPR faults. Therefore, the fault coverages that can be obtained with STUMPS BIST are often low for circuits which suffer from many RPR faults.

   The fault coverage can be increased by applying more PR test patterns, but more test patterns means longer test times and, given a reasonable number of test patterns/test time, the obtained fault coverages are still too low. However, the insertion of TPs to improve the PR detectabilities in the circuit will result in an increased fault coverage.

2. *Lower fault coverage caused by correlation between the patterns shifted into the different scan-chains*
   LFSRs which feed multiple scan chains in parallel can result in lower fault coverages due to structural dependencies introduced by the TPG [Raj99]. When the scan-chains are fed directly from adjacent stages in the LFSR, the chains will contain highly correlated patterns; the difference between the patterns is that they are only shifted one bit. For example, the first chain will contain the pattern 1*0011001* while the second one contains pattern *0011001*0.

   This problem can be solved by adding a Phase Shifter [Raj99] (or XOR cloud) between the LFSR/CA and the scan-chains.

3. *Unknown MISR states caused by unknown/fixed input values*
   It is possible that several inputs of the CUT have an unknown/fixed value, e.g., inputs from an embedded memory. These unknown values result in paths with unknowns in the CUT and can finally propagate to one or several outputs (the propagation of unknowns is also called X-path propagation). When an unknown value is captured into a MISR, the complete state of the MISR cannot be determined and with an unknown MISR state, it cannot be checked whether a CUT is fault-free or not.

   This problem can be solved by the insertion of additional logic/TPs such that, in test mode, these unknown values are replaced by known, fixed or PR values. Another solution is to insert additional logic at the outputs which capture unknown states and map them to known states.

4. *Possible circuit damage and unknown MISR states caused by bus-conflicts*
   With ATPG one can avoid bus-conflicts. The ATPG just generates patterns such that all buses are only driven by at most one driver. But this is not the case for PR patterns; it is possible that some buses are driven by multiple drivers. This results in bus-conflicts that can damage the circuit. These conflicts also result in unknown values that can propagate into the MISR, corrupting the MISR state.

There are three general solutions for this problem:

(a) Insert additional logic in the circuit such that for all patterns, no bus is driven by multiple drivers at the same time.

(b) Insert additional logic between the PRTPG and the original circuit such that only patterns are shifted into the scan-chains that will not result in bus-conflicts.

(c) Do not prevent the bus-conflicts (can result in circuit-damage), but insert additional logic between the outputs of the circuit and the MISR such that all possible unknowns are mapped into known values such that the MISR state is known for the fault-free case.

3.7 State-of-the-art TPI algorithms

In this section, several state-of-the-art TPI algorithms will be described. Subsection 3.7.1 describes the *Cost Reduction Factor (CRF)* algorithm. This algorithm calculates for each TP candidate an estimate for the reduction in the global test cost of the circuit. Subsection 3.7.2 describes an algorithm which calculates a more accurate estimate for the reduction in cost for each TP candidate; the *Hybrid Cost Reduction Factor (HCRF)*. Subsection 3.7.3 describes a TPI algorithm that splits the test into multiple phases. In each phase a different set of TPs will be enabled. Subsection 3.7.4 gives a short description of several other TPI algorithms found in literature.

3.7.1 The Cost Reduction Factor (CRF) TPI algorithm

In [Sei91], an O(n) algorithm is described which calculates an estimate for the reduction in cost that a TP candidate would cause. This estimate is called the *Cost Reduction Factor (CRF)*. The CRF for a given signal line that will be TP candidate, is calculated by using only local testability information of that line, i.e., the COP controllability, observability and gradient values. Therefore the CRF for all TP candidates can be calculated very fast, while the calculation of the *Actual Cost Reduction Factor (ACRF)* for all TP candidates would be very CPU-consuming. The ACRF of a TP candidate is obtained by temporarily inserting the TP into the circuit, re-evaluating the CF and determining the ACRF value by subtracting the new cost value from the old one. This is an O(n^2) algorithm, i.e., for each *n* TP candidates (assuming all lines in the circuit can be TP candidate) in the circuit, a TP is inserted after which for all *n* lines in the circuit the cost is recalculated.

In Seiss et al. [Sei91], the CRFs for AND-gate CPs, OR-gate CPs and OPs are derived. The CRFs for a CP at line *l* for an AND respectively and OR gate are given in Eqs. 3.23 and 3.24.

$$CRF_l^{AND} = \frac{1-C_l}{2-C_l} \cdot C_l \cdot \frac{dK}{dC_l} - \frac{2}{C_l \cdot (1-C_l) \cdot W_l} \tag{3.23}$$

$$CRF_l^{OR} = \frac{C_l - 1}{C_l + 1} \cdot C_l \cdot \frac{dK}{dC_l} - \frac{2}{C_l \cdot (1 - C_l) \cdot W_l} \tag{3.24}$$

The CRF for an OP at line *l*, derived in [Sei91], is given in Eq. 3.25:

$$\begin{aligned} CRF_l^{OP} &= (W_l - 1) \cdot W_l \cdot \frac{dK}{dW_l} \\ &\quad \text{if line } l \text{ represents a fan-out stem in the original circuit} \\ CRF_l^{OP} &= (W_l - 1) \cdot W_l \cdot \frac{dK}{dW_l} - \left(\frac{1}{C_l} + \frac{1}{1 - C_l} \right) \\ &\quad \text{otherwise} \end{aligned} \tag{3.25}$$

The distinction between line *l* representing a fan-out stem or not can be explained by considering Fig. 3.4.a and 3.4.b. In the case that line *l* is not a fan-out stem in the original circuit, the introduction of the additional fan-out branch, which is required to implement the OP, causes line *l* to become a fan-out stem in the modified circuit. As a consequence, two new faults have to be modeled, which are the SA0 and SA1 faults for the newly created fan-out stem. Since these two faults are observable on the new PO with observability 1, their contribution to the cost and CRF_l^{Obs} is given by $1/C_l + 1/(1\text{-}C_l)$. When *l* already denotes a fan-out stem in the original circuit, no new faults have to be modeled.

The CRFs, however, could differ significantly from the ACRFs, for the following reasons:

1. The observability changes due to a CP are completely ignored. This means that the observability changes of lines in the fan-in cone of POs, reachable from the CP, are completely neglected.

2. Certain circuit structures and fault sets were assumed during the derivation of the CRF equations. By assuming that CUTs contain these types of structures, a controllability value C_l can be factored out such that the rest of the terms in K are completely independent to C_l. This assumption is not generally true.

The *CRF TPI* algorithm of Seiss et al. [Sei91] is listed in Algorithm 3.2. In each iteration of the outermost while loop, the algorithm inserts exactly one CP or OP into the circuit. TPs are inserted until the cost K of the circuit reaches a desired cost ($K_{desired}$), or the number of inserted TPs (*#TPs*) has reached the maximum number of TPs ($\#TP_{max}$) that are allowed to be inserted. Because the controllability, observability, controllability and observability gradients, and the CRFs may have changed due to the insertion of the TP, these values have to be updated at the beginning of each iteration. Subsequently, the algorithm selects a set of candidate lines for TPI by exploiting the CRFs *as a guidance*. The CRFs are not very accurate and as described in [Tsa97], the inaccuracy of the CRFs becomes more significant when a high fault coverage is reached. Therefore, the CRF is only used to pre-select a small number of possible TPs in the CRF TPI algorithm. The ACRFs are only calculated for the pre-selected lines. This saves a lot of time because

Algorithm 3.2 CRF Test Point Insertion

Input: CUT, desired cost ($\boldsymbol{K}_{desired}$), $\boldsymbol{TP}_{max}$

Output: CUT with TPs

```
while ( $K > K_{desired}$) and ($\#TPs < \#TP_{max}$) do
  for all lines $l \in$ CUT do
    Evaluate controllability $C_l$ and observability $W_l$
    Compute the gradients d$K$/d$W_l$ and d$K$/d$C_l$
    Compute $CRF_l^{OP}$
    if (d$K$/d$C_l < 0$) then
      Compute $CRF_l^{OR}$
    else
      Compute $CRF_l^{AND}$
  Determine $CRF^{CP_{max}}$ with $\forall_{l \in \text{CUT}}(CRF_l^{AND/OR} <= CRF^{CP_{max}})$
  Determine $CRF^{OP_{max}}$ with $\forall_{l \in \text{CUT}}(CRF_l^{OP} <= CRF^{OP_{max}})$
  Determine the set $SL_{CP}$ of all lines $l \in$ CUT with
     $SL_{CP} = \{l \mid CRF_l^{AND/OR} > 0.1 \cdot CRF^{CP_{max}}\}$
  Determine the set $SL_{OP}$ of all lines $l \in$ CUT with
     $SL_{OP} = \{l \mid CRF_l^{OP} > 0.1 \cdot CRF^{OP_{max}}\}$
  $NR = 1$
  for all lines $l \in SL_{CP}$ do
    if (d$K$/d$C_l < 0$) then
      Simulation of an OR type CP at line $l$
    else
      Simulation of an AND type CP at line $l$
    Evaluation of cost function $K^m$ for the modified CUT
    $cost[NR] = K^m$
    $NR = NR + 1$
  for all lines $l \in SL_{OP}$ do
    Simulation of an OP at line $l$
    Evaluation of cost function $K^m$ for the modified CUT
    $cost[NR] = K^m$
    $NR = NR + 1$
  Select line $l_{TP}$ with $\forall_{0 < l <= NR}(cost[l]) >= cost[l_{TP}])$
  if ($l_{TP} \in SL_{CP}$) then
    if (d$K$/d$C_{l_{TP}} < 0$) then
      Insert an OR type CP at line $l_{TP}$
    else
      Insert an AND type CP at line $l_{TP}$
  else
    Insert an OP at line $l_{TP}$
```

calculation of the more accurate ACRFs is very CPU time-consuming and therefore not practical for current large & complex circuits, e.g., calculating the ACRFs for all TP candidate lines in circuit c7552 takes 625 seconds on a 700 MHz AMD Athlon, while calculating the CRFs takes only 0.01 seconds. Therefore only lines with CRFs that are at least 10% (the 0.1 factor in Algorithm 3.2) of the highest CRF found in the circuit are considered TP candidate. Each TP candidate is temporarily inserted to determine the COP testability measures and the value of the CF K_m for the modified circuit such that the ACRF can be determined. Once all candidates have been examined, the candidate CP or OP which leads to the largest cost reduction, is actually inserted in the circuit.

The 10% factor can be considered as a parameter with which the algorithm can be tuned. A higher percentage than 10% means that there will be fewer TP candidates, speeding up the algorithm. However, this might also result in a less accurate TPI; good TP positions are not considered TP candidate due to their inaccurate low CRF values. A lower percentage means more TP candidates, resulting in reduced performance but possibly more accurate TPI because more TPs candidates are evaluated, including good TP positions with an inaccurate low CRF value.

Experimental results of the CRF TPI Algorithm

[Sei91] gives experimental results of the CRF TPI algorithm for several ISCAS benchmark circuits [Brg85][Brg89], and industrial circuits, which are known to be highly RPR. They are shown in Table 3.2. Column *Circuit* gives the name of the circuit and Column *#Gates* lists the number of gates in the circuit. Columns *FC before TPI* and *FC after TPI* show the fault coverages achieved after the simulation of 32,000 PR patterns for the circuit without TPs, respectively after TPI. All redundant faults have been identified in advance and are excluded from the fault list. Therefore, all fault coverages refer to non-redundant faults only. Column *#TP* shows the number of inserted TPs and Column *#CP/#OP* shows how the inserted TPs are divided between CPs and OPs. Column *Est. area overhead* shows an estimation of the overhead of the TPs compared to the circuit itself.

The results from Table 3.2 show that the insertion of a small number of TPs leads to

Table 3.2: Results of the CRF TPI algorithm upon RPR circuits

Circuit	#Gates	FC before TPI	FC after TPI	#TP	#CP / #OP	Est. area overhead
s838	422	87.51 %	100.00 %	3	3/0	1.54 %
c2670	1193	88.21 %	100.00 %	10	3/7	1.21 %
c7552	3512	96.11 %	100.00 %	20	18/2	1.26 %
BNR_A	16143	96.86 %	99.40 %	10	9/1	0.15 %
BNR_B	13019	92.86 %	98.14 %	20	13/7	0.39 %
BNR_C	20900	90.35 %	99.41 %	20	20/0	0.27 %

Table 3.3: Estimated detection probabilities of the hardest-to-detect faults end the test length required to achieve 99% fault coverage, *before* and *after* TPI.

Circuit	Pd_f hardest-to-detect faults before TPI	Pd_f hardest-to-detect faults after TPI	Test length 99 % FC before TPI	Test length 99 % FC after TPI
s838	$3.36\cdot10^{-11}$	$2.57\cdot10^{-04}$	$>$ 10 000 000	2 000
c2670	$3.47\cdot10^{-06}$	$2.38\cdot10^{-04}$	420000	1 200
c7552	$1.42\cdot10^{-13}$	$4.26\cdot10^{-05}$	8900000	1 800
BNR_A	$6.03\cdot10^{-11}$	$1.33\cdot10^{-06}$	$>$ 10000000	21 000
BNR_B	$1.86\cdot10^{-09}$	$5.27\cdot10^{-06}$	6 400000	36 000
BNR_C	$2.16\cdot10^{-19}$	$2.48\cdot10^{-06}$	$>$ 10000000	7 300

a significant increase of the fault coverage achievable with 32,000 random patterns, e.g., by inserting only 3 TPs in circuit s838, the fault coverage has increased with 13% from 87% to 100%. For the remaining circuits, the increase in fault coverage ranges from 4% to 12%, after the insertion of only 10 to 20 TPs. The results also show that the estimated area overhead is in the order of 1% for all circuits, and even smaller for the larger, industrial circuits. The area-overhead estimates do take in account the additional SFFs, which have to be inserted in the scan-chain in order to feed the CPs and to capture the test responses at the OPs.

Because they are good indicators for the random pattern testability of a circuit, the detection probabilities of the hardest-to-detect faults for the circuits listed in Table 3.2 are given in Table 3.3. Another criterion characterizing the random pattern testability of a circuit consists of the test length required to achieve a certain fault coverage. Table 3.3 also lists the test length which is necessary to achieve a fault coverage of 99%. Columns *Pd_f hardest-to-detect faults before TPI*, respectively *after TPI*, show the lowest detection probability found in the circuit before TPI, respectively after TPI. Columns *Test length 99% FC before TPI*, respectively *after TPI*, show the number PR patterns that are applied until the 99% fault coverage level has been reached before TPs, respectively after TPs, have been inserted. The results show that the detection probability of the hardest-to-detect faults in the modified circuits are improved by several orders of magnitude, compared to the ones of the original circuits; e.g., the detection probability of the hardest-to-detect fault in circuit c7552 has been improved by 8 orders of magnitude from $1.42\cdot10^{-13}$ to $4.26\cdot10^{-5}$. In circuit BNR_C the improvement even is 13 orders of magnitude, from $2.16\cdot10^{-19}$ to $2.48\cdot10^{-6}$.

Before TPI, all circuits listed in Table 3.3 require far more than 32,000 PR patterns before the 99% fault coverage level is reached, ranging from 420,000 patterns for circuit c2670 to more than 10 million patterns for circuits s838, BNR_A and BNR_C. After TPI, all circuits can be tested with less than 32,000 PR patterns, except circuit BNR_B which requires 36,000 PR patterns. But even the test length for circuit BNR_B after TPI is only 36,000/6,400,000=0.56% of the test length required without TPs. The test time is proportional with the test length, hence the same reduction in test time is achieved by the TPI.

That the CRF TPI algorithm can be used in combination with a BIST method, i.e., bit-flipping BIST [Kie98] described in Subsection 2.7.3, to facilitate the BIST method in reaching higher fault coverages with less silicon overhead, is shown in [Vra02].

CRF TPI derived algorithms

In Cheng et al. [Che95b], the CRF TPI algorithm has been adjusted such that it can also be applied to partial-scan circuits, i.e., circuits that contain both scannable as non-scannable flipflops. The CRF based TPI algorithm of [Che95b] also takes into account timing information, such that TPs are not inserted at critical paths.

In Nakao et al. [Nak99], also a CRF based TPI algorithm for BIST is presented. In order to reduce the delay of the TPI, TPs (CPs) are only inserted at inverters and buffer positions. Buffers/inverters are replaced by AND/OR gates controlled by SFFs which must provide a non-controlling value during normal application mode. Also the OPs are implemented by means of SFFs. If possible, SFFs that control or observe the TPs, are shared in order to minimize the hardware overhead.

3.7.2 The Hybrid Cost Reduction Factor (HCRF) TPI algorithm

This section describes a TPI algorithm presented by Tsai et al. [Tsa97]. It uses the *Hybrid Cost Reduction Factor (HCRF)* for estimating the Actual Cost Reduction Factor. The key idea of the HCRF TPI algorithm is that an *event-driven mechanism* is used to propagate the testability changes due to a TP. Once the changes become small, the gradients (see Section 3.5.3) can be used to estimate the testability changes of all other lines to avoid high computational complexity of recalculating the testabilities for the entire circuit.

The CRFs can deviate significantly from the ACRFs, especially when the circuit has a high fault coverage [Tsa97]. Therefore, the CRF Test Point Selection algorithm presented in the previous section only uses the CRF values as a guidance to select the best TP to insert. However, the ACRF values still have to be calculated for the limited set of possible TPs, which remains after the pre-selection based on the highest CRF values. When the set of candidate TPs is too small, the number of calculations and the CPU time necessary is limited, but it is very well possible that several good TPs are excluded because their CRF values are not accurate. On the other hand, a larger set of candidate TPs means more calculations and possibly impractical CPU consumption.

With the HCRF TPI algorithm, it is not necessary to do a pre-selection on all TP candidates and to calculate the ACRF for all pre-selected candidates, by temporarily inserting the TP. Only the HCRF estimates from all TP candidates are used to select the TP that will be inserted. Before the HCRF calculation, and the TPI algorithm based on the HCRF, are explained, first the relation between the ACRF and the HCRF is given.

Given a fault set *F*, the *ACRF* for a TP can be expressed by:

$$\begin{aligned} ACRF &= K^{(Org)} - K^{(m)} \\ &= \sum_{f \in F} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right) \\ &= \underbrace{\sum_{f \in F_1} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)}_{\text{the difference is large}} + \underbrace{\sum_{f \in F_2} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)}_{\text{the difference is small}} \end{aligned} \tag{3.26}$$

where $K^{(Org)}$, $K^{(m)}$ are the cost values, and $Pd_f{}^{(Org)}$, $Pd_f{}^{(m)}$ the detection probabilities of fault *f*, before, respectively after, the modification of the circuit by the inserted TP (candidate). The fault set F can be divided into two subsets, F_1 and F_2. For each fault in F_1, $\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}}$ is large and therefore the gradient approximation is not accurate. On the other hand, for every fault in F_2, the difference of $\frac{1}{Pd_f{}^{(Org)}}$ and $\frac{1}{Pd_f{}^{(m)}}$ is small enough such that the gradients can accurately enough estimate their impact on K, see Subsection 3.5.3.

Typically, the size of F_1 is much smaller than that of F_2, because the test only influences a small part of the circuit. Therefore, $\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}}$ can be calculated explicitly for faults in F_1 (by computing $\frac{1}{Pd_f{}^{(m)}}$) and the gradients can be used to evaluate the second term of Eq. 3.26. An event-driven mechanism is used to propagate the controllability/observability changes caused by the TP candidate until the changes drop below a threshold. Whether an event at line l_{F_1} is scheduled for further propagation is determined by the ratio of $dK/dW_{l_{F_1}} \cdot \Delta W_{l_{F_1}}$ to $K^{(Org)}$. The change in observability of line l_{F_1} is represented by $\Delta W_{l_{F_1}}$. The region of lines processed to identify set F_1 is relatively very small for a large design, therefore the complexity for calculating the first term of Eq. 3.26 is low. In addition, gradients can provide a very good estimate for the second term of Eq. 3.26, because the assumptions of the gradients are not violated for faults in F_2. How to calculate the HCRF for OPs, respectively CPs, is given in the following text.

Calculation of the HCRF for OPs

The propagation of the explicit recalculation of the COP observability changes of fault set F_1 starts from the OP l backward to the PIs, shown in Fig. 3.10. For each gate output z_{gate} inside Region I, the gradient approximation, i.e., the ratio of $\mathrm{d}K/\mathrm{d}W_{z_{gate}} \cdot \Delta W_{z_{gate}}$ to $K^{(Org)}$, is larger than a given (user-defined) threshold. As a result, the event-driven mechanism continues with the explicit recalculation of the COP observabilities of the gate inputs. Propagation of the explicit recalculation stops when the gradient approximation becomes below the user-defined threshold, illustrated in Fig. 3.10 by the Boundary line.

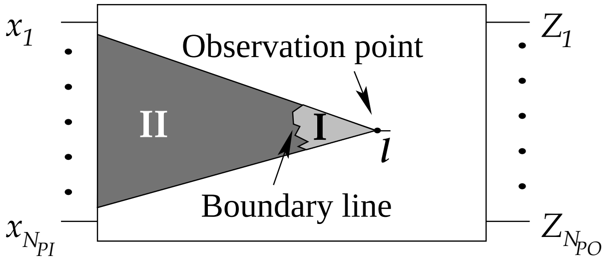


Figure 3.10: Computing HCRF for an OP

At the Boundary line, the gradient approximation is used to estimate the cost impact of the observability changes of the faults in Region II.

The HCRF for an OP comprises three parts:

1. $\sum_f \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)$ for every fault f inside Region I; the new observabilities and therefore the new Pd_f*'s* are computed explicitly.
2. For all faults inside Region II, $\sum_{l_{bound}} \frac{dK}{dW_{l_{bound}}} \cdot \Delta W_{l_{bound}}$, where l_{bound}'s are the lines *on the boundary* in Fig. 3.10, is used to estimate their contribution to the ACRF.
3. If the OP l is not a fan-out stem in the original circuit, the contribution of the new fan-out branch faults ($\frac{1}{Pd_{l/SA1}}$ and $\frac{1}{Pd_{l/SA0}}$) are added.

The equation to compute the HCRF for an OP becomes:

$$HCRF_l^{OP} = \sum_{f \in Region\ I} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right) - \sum_{l_b \in Boundary} \left(\frac{dK}{dW_{l_b}} \cdot \Delta W_{l_b} \right) \quad \text{when line } l \text{ is a fan-out stem} \tag{3.27}$$

$$HCRF_l^{OP} = \sum_{f \in Region\ I} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right) - \sum_{l_b \in Boundary} \left(\frac{dK}{dW_{l_b}} \cdot \Delta W_{l_b} \right) - \left(\frac{1}{Pd_{l/SA0}} + \frac{1}{Pd_{l/SA1}} \right) \quad \text{when line } l \text{ is not a fan-out stem} \tag{3.28}$$

The cost contribution estimate for the lines in Region II can be viewed as adding a set of pseudo-OPs (with a delta change in observability) on the boundary simultaneously. The second term of Eq. 3.27 simply represents the superposition of the effects of all these pseudo-OPs. Because the change is small, the assumption of the gradients holds and therefore the accuracy of this term will be high enough.

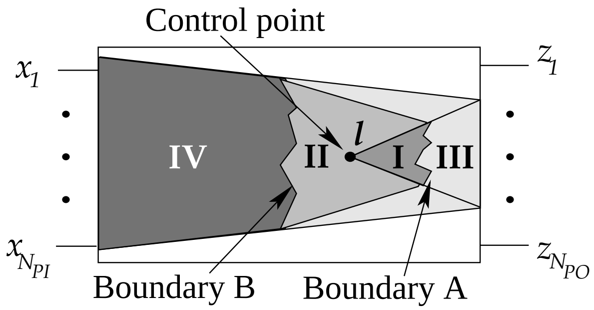


Figure 3.11: Computing HCRF for an AND/OR CP

Calculation of the HCRF for CPs

Computing the HCRF for a CP is more complicated than for an OP. A CP does not only change the controllabilities, but also the observabilities in the circuit. The propagation of the altered COP values has to proceed in both forward and backward directions. Starting from the CP, the propagation of the new controllabilities proceeds forward to the POs shown as Region I in Fig. 3.11. During the processing of each gate input x_{gate}, the ratio of $\mathrm{d}K/\mathrm{d}C_{x_{gate}} \cdot \Delta \mathrm{C}_{x_{gate}}$ to $K^{(Org)}$ is compared with a given user-defined threshold to decide whether the controllabilities of the gate outputs should be recalculated or that the controllability gradient can be used to estimate the impact on the cost.

Once the forward propagation ends, a set of lines is obtained; indicated as Boundary A in Fig. 3.11. The direction of propagations is then changed to backward to the PIs. The lines on Boundary A are used as starting points for this backward propagation, similar to the HCRF for OP calculation. Again, when the ratio of $\mathrm{d}K/\mathrm{d}W_{z_{gate}} \cdot \Delta \mathrm{W}_{z_{gate}}$ to $K^{(Org)}$ becomes small, the backward propagations stop, and a set of lines indicated as Boundary B in Fig. 3.11 is identified.

The HCRF of a CP contains four terms:

1. For every fault f inside Regions I and II, $\sum_f \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)$ is computed explicitly. New controllabilities and observabilities and therefore Pd_f*'s* are computed by the above mentioned event-driven procedures.

2. $\sum_{l_{bA} \in boundary\ A} \left(\frac{dK}{dC_{l_{bA}}} \cdot \Delta C_{l_{bA}} \right)$ is used to estimate $\sum_{f \in Region\ III} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)$.

3. $\sum_{l_{bB} \in boundary\ B} \left(\frac{dK}{dW_{l_{bB}}} \cdot \Delta W_{l_{bB}} \right)$ is used to estimate $\sum_{f \in Region\ IV} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)$.

4. The contribution of the new faults introduced by the inserted gate are added: $\frac{1}{Pd_{l/SA0(SA1)}}$ and $\frac{1}{Pd_{t/SA0(SA1)}}$. These are the SA0 (SA1) faults on the extra input and the output of the added OR(AND) gate.

The equations to compute the HCRF for OR-type and AND-type CPs become:

$$\begin{aligned} HCRF_l^{OR} &= \sum_{f\in Regions\ I\&II}\left(\frac{1}{Pd_f{}^{(Org)}}-\frac{1}{Pd_f{}^{(m)}}\right)-\sum_{l_{bA}\in boundary\ A}\left(\frac{dK}{dC_{l_{bA}}}\cdot\Delta C_{l_{bA}}\right) \\ &\quad -\sum_{l_{bB}\in boundary\ B}\left(\frac{dK}{dW_{l_{bB}}}\cdot\Delta W_{l_{bB}}\right)-\left(\frac{1}{Pd_{l/SA0}}+\frac{1}{Pd_{t/SA0}}\right) \qquad (3.29) \\ HCRF_l^{AND} &= \sum_{f\in Regions\ I\&II}\left(\frac{1}{Pd_f{}^{(Org)}}-\frac{1}{Pd_f{}^{(m)}}\right)-\sum_{l_{bA}\in boundary\ A}\left(\frac{dK}{dC_{l_{bA}}}\cdot\Delta C_{l_{bA}}\right) \\ &\quad -\sum_{l_{bB}\in boundary\ B}\left(\frac{dK}{dW_{l_{bB}}}\cdot\Delta W_{l_{bB}}\right)-\left(\frac{1}{Pd_{l/SA1}}+\frac{1}{Pd_{t/SA1}}\right) \qquad (3.30) \end{aligned}$$

Every line l_{bB} on Boundary B can be viewed as a pseudo-OP such that the superposition $\sum_{l_{bB}}(\frac{dK}{dW_{l_{bB}}}\cdot\Delta W_{l_{bB}})$ can be used to estimate $\sum_f\left(\frac{1}{Pd_f{}^{(Org)}}-\frac{1}{Pd_f{}^{(m)}}\right)$ for faults inside Region IV. Similarly, every line l_{bA} on Boundary A can be viewed as a pseudo-CP and $\sum_{l_{bA}}(\frac{dK}{dC_{l_{bA}}}\cdot\Delta C_{l_{bA}})$ is used to estimate the contribution for faults inside Region III. A CP however, affects the testabilities of both its fan-ins and fan-outs. Inserting a set of CPs on Boundary A should have effects on all faults in Regions I-IV. This means $\sum_{l_{bA}\in boundary\ A}(\frac{dK}{dC_{l_{bA}}}\cdot\Delta C_{l_{bA}})$ also contains contributions from Region I, II and IV. Because the contribution from Region III usually dominates $\frac{dK}{dC_{l_{bA}}}\cdot\Delta C_{l_{bA}}$, the error by this approximation is negligible.

Furthermore, unlike OPs which always reduce K when they are inserted simultaneously, the effects of CPs can cancel one another if they are added at the same time. For these reasons, using $\sum_{l_{bA}\in boundary\ A}(\frac{dK}{dC_{l_{bA}}}\cdot\Delta C_{l_{bA}})$ to estimate $\sum_f\left(\frac{1}{Pd_f{}^{(Org)}}-\frac{1}{Pd_f{}^{(m)}}\right)$ for the faults f is just a heuristic.

The Hybrid CRF TPI Algorithm

The Hybrid CRF TPI Algorithm is listed in Algorithm 3.3. Instead of pre-selecting a set of TP candidates that is further analyzed by calculating their ACRFs, for *all* lines the HCRF for a CP and the HCRF for an OP are calculated. Given all L lines in the circuit, the line l_{CP} with the highest HCRF for AND/OR CPs is selected. Also the line l_{OP} with the highest HCRF for OPs is selected. When the highest HCRF for CPs is larger than the highest HCRF for OPs, a CP is inserted at line l_{CP}, otherwise an OP is inserted at line l_{OP}.

In the Hybrid CRF TPI algorithm it is assumed that for *all* lines in the circuit the HCRFs are calculated. There are of course lines in the circuit which can be excluded as TP candidate in advance. All inputs (PIs/POs and SFF in/outputs) can be excluded as TP candidate. When timing information is available, i.e., critical paths in the circuit are known, also the lines at which a TP would result in a performance degradation, i.e., would increase the critical path length, can be excluded as TP candidate.

Algorithm 3.3 Hybrid CRF Test Point Insertion

Input: CUT, desired cost ($\boldsymbol{K}_{desired}$), $\boldsymbol{TP}_{max}$

Output: CUT with TPs

```
while ( $K > K_{desired}$) and ($\#TPs < \#TP_{max}$) do
  for all lines $l \in$ CUT do
    Evaluate controllability $C_l$ and observability $W_l$
    Compute the gradients $dK/dW_l$ and $dK/dC_l$
    Compute $HCRF_l^{OP}$
    if ($dK/dC_l < 0$) then
      Compute $HCRF_l^{OR}$
    else
      Compute $HCRF_l^{AND}$
  Select line $l_{CP}$ with $\forall_{l \in \mathrm{CUT}}(HCRF_l^{AND/OR} <= HCRF^{CP_{max}})$
  Select line $l_{OP}$ with $\forall_{l \in \mathrm{CUT}}(HCRF_l^{OP} <= HCRF^{OP_{max}})$
  if ($HCRF^{CP_{max}} >= HCRF^{OP_{max}}$) then
    if ($dK/dC_{l_{CP}} < 0$) then
      Insert an OR type CP at line $l_{CP}$
    else
      Insert an AND type CP at line $l_{CP}$
  else
    Insert an OP at line $l_{OP}$
```

Experimental results of the Hybrid TPI algorithm

Tables 3.4 and 3.5 show figures, taken from [Tsa97], representing the quality of the TPs selected by the Hybrid TPI algorithm, respectively the CRF TPI algorithm described in Subsection 3.7.1. Twenty TPs are selected sequentially for the ISCAS'85 circuits c2670 and c7552. At each iteration, the ACRFs for *all possible* TPs are computed explicitly and sorted in a descending order. Then the ranks of the TPs selected by the Hybrid TPI algorithm and by the CRF TPI algorithm are checked. The columns in Tables 3.4 and 3.5 show the rank of the TPs selected by the HCRF, respectively CRF TPI algorithm, for each iteration.

The results show that the TPs selected by the Hybrid TPI algorithm are almost identical to the ones with the highest ACRFs. Except for the nineteenth iteration, the HCRF selects the TP with the highest ACRF for circuit c2670. At the nineteenth iteration it still selects the second best. During the first nine iterations for circuit c7552, the HCRF selects the TP candidate with the highest ACRF, at the tenth iteration, the 7th best ACRF candidate is chosen and between iterations 11 and 20 all TP candidates are in the top 3.

The rank of the TPs selected by the CRF TPI algorithm is sometimes quite off the highest ranked ACRF candidate, especially for circuit c2670. Between the tenth and twentieth iteration, the rank of the CRF TP is between 1991 and 1925 of approximately

Table 3.4: Rankings of TPs selected by the Hybrid TPI algorithm

Circuit	Iteration																			
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th	16th	17th	18th	19th	20th
c2670	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1
c7552	1	1	1	1	1	1	1	1	1	7	1	1	1	2	2	1	3	2	1	1

Table 3.5: Rankings of TPs selected by the CRF TPI algorithm

Circuit	Iteration																			
	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	12th	13th	14th	15th	16th	17th	18th	19th	20th
c2670	106	8	4	6	1	3	1	1	1	1881	1881	1885	1895	11	1901	1901	1911	1	1925	1925
c7552	1	1	11	1	1	1	1	1	1	29	1	1	3	2	1	1	1	2	1	1

2670 TP candidates. The CRF TP ranks for circuit c7552 are not so far off and except for iterations three and ten, all ranks are in the top 3.

Whether the more accurate TP selection of the Hybrid TPI algorithm also results in higher fault coverages is shown in Table 3.6. This table shows results taken from [Tsa97] of the Hybrid TPI algorithm and the CRF TPI algorithm on eight, RPR, ISCAS'85 and ISCAS'89 benchmark circuits. The circuits have been first optimized for performance using Berkeley SIS synthesis system [Sen92] and are mapped into Lucent 0.9 micron CMOS cell library. In this experiment, the threshold of scheduling an event for propagation or using the gradients to estimate the impact on the cost, is set to 0.1% for HCRF^{OBS}, for HCRF^{OR} and for HCRF^{AND}. The fault coverages shown are computed after applying 32,000 PR patterns. For both TPI algorithms, the same number of TPs are inserted, listed in Column *#TP*. The fault coverages and CPU times are shown in Columns *FC(%)* and *CPU time* for respectively the Hybrid TPI algorithm and the CRF TPI algorithm. Column *CPU time ratio(%)* shows the ratio of the CPU time of the Hybrid TPI algorithm to that of the CRF TPI algorithm. The CPU time is measured on a SUN SPARCstation 5.

The results show that indeed the Hybrid TPI algorithm results in the same (circuits c2670, s3384) or higher fault coverages (the other six listed circuits) than the CRF TPI

Table 3.6: Comparison between the Hybrid and the CRF TPI algorithm

		Hybrid		CRF		CPU time
Circuit	#TP	FC(%)	CPU time	FC(%)	CPU time	ratio(%)
c2670	5	98.31	78.0 sec.	98.31	264.0 sec.	30
c7552	10	98.23	3.4 min.	97.92	22.0 min.	15
s3330	13	99.92	4.9 min.	99.39	29.0 min.	17
s3384	9	99.78	128.0 sec.	99.78	659.0 sec.	19
s4863	2	99.64	61.0 sec.	99.61	201.0 sec.	30
s9234	19	96.10	12.3 min	94.79	75.0 min.	16
s15850	34	97.41	34.4 min.	97.08	7.7 hr.	7
s38417	46	99.19	1.7 hr.	98.84	92.1 hr.	2
$\sum$	138	98.49	2.7 hr.	98.08	102.2 hr.	3

algorithm. The last line in Table 3.6 shows the average fault coverages achieved after TPI for both algorithms. These averages are calculated taking into account the circuit size. The CRF TPI algorithm reaches an average of 98% fault coverage. 2% of all faults are not covered by the 32,000 PR patterns. On average, the Hybrid TPI algorithm only misses 1.5% of the faults and covers 98.5%.

The CPU time ratio figures show that the Hybrid TPI algorithm runs faster than the CRF TPI algorithm. Only 2% (for circuit s38417) up to 30% (for circuits c2670 and s4863) of the time that the CRF TPI algorithm takes, is spent by the Hybrid CRF algorithm. The last line shows the total time spent on TPI for both algorithms. The Hybrid TPI algorithm only takes 3% of the time that the CRF TPI algorithm consumes. Especially the TPI runs on the two largest of the eight circuits (s15850, s38417) are much faster with the Hybrid TPI algorithm and cause this large speed difference. These results show that the Hybrid TPI algorithm is more feasible on larger designs than the CRF TPI algorithm. This large reduction of the CPU time for the Hybrid TPI algorithm mainly comes from the elimination of the second step in the CRF TPI algorithm, the calculation of the ACRFs.

3.7.3 Multi-phase Test Point Insertion

The Multi-phase TPI (MTPI) algorithm [Tam96] is based on a constructive methodology. A divide and conquer technique is used to partition the entire PR test set into multiple phases. In each phase a group of TPs (both CPs and OPs), targeting a specific set of faults, is selected. Within a particular phase of the test, one group of CPs is enabled and the other CPs are disabled. This remains the same during the whole phase, i.e., during a particular phase the CPs are controlled by fixed values. On the other hand, the CPs selected by the (H)CRF TPI algorithm are driven by independent, equi-probable signals instead of fixed ones, and test application is performed in a single session by enabling all CPs

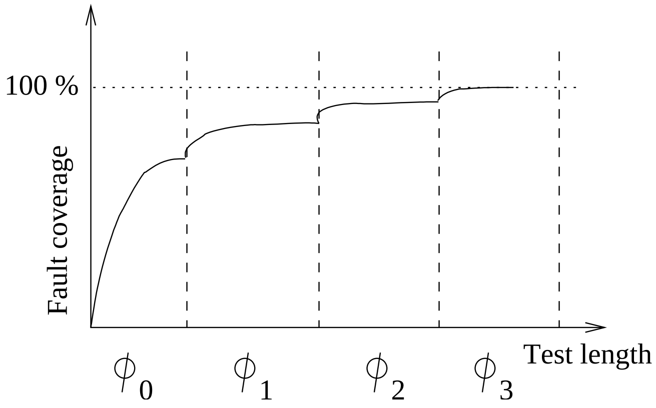


Figure 3.12: FC plot of a circuit which test is divided into 4 phases

(equi-probable) and OPs simultaneously. Because CPs can be enabled simultaneously, it is possible that they have conflicting values such that some faults remain undetected. With MTPI this can be avoided by enabling possible conflicting CPs in different phases.

An example of the division into phases is given in Fig. 3.12. As is shown, each phase contributes to the results achieved so far, moving the solution closer to complete fault coverage. The design of each phase, i.e., the selection of CPs and OPs, is guided by progressively reducing the set of undetected faults. Within each phase, CPs are identified which maximally contribute to the fault coverage achieved so far. A probabilistic fault simulation technique is used to compute the impact of a new CP in the presence of the CPs selected so far.

The MTPI BIST scheme and methodology

Fig. 3.13 illustrates the main components of the MTPI BIST scheme, which uses the STUMPS approach.[2] The *BIST controller* contains a *Pattern counter* and a *Phase decoder*. The phase decoder plays a key role in the MTPI BIST scheme. The pattern counter is used as input for the phase decoder to know in which phase the test is. The outputs of the phase decoder are used to enable the group of CPs belonging to the current phase.

In Fig. 3.13, the test T has been divided into four phases, e.g., $\phi_0 \ldots \phi_3$. The corresponding phase decoder output values during the test are shown in Fig. 3.14. In the first phase ϕ_0, the first quarter of the test, all CPs are disabled: $a' = a$ and $b' = b$. During phase ϕ_1 ($\phi_1 = 1$, $\phi_2 = \phi_3 = 0$), the AND-type CP CP_a is enabled, forcing line a' to 0, while the OR-type CP CP_b is disabled; the value of line b' equals the value b. During phase ϕ_2 both CPs CP_a and CP_b are enabled, forcing line a' to 0 and line b' to 1. Finally, in phase ϕ_3 only CP CP_b is enabled, line b' is forced to 1 while the value of line a is propagated unchanged to line a'.

---

[2] The scan-chains, including the surrounding scan-chains are still present in MTPI BIST, however not shown to keep Fig. 3.13 clear.

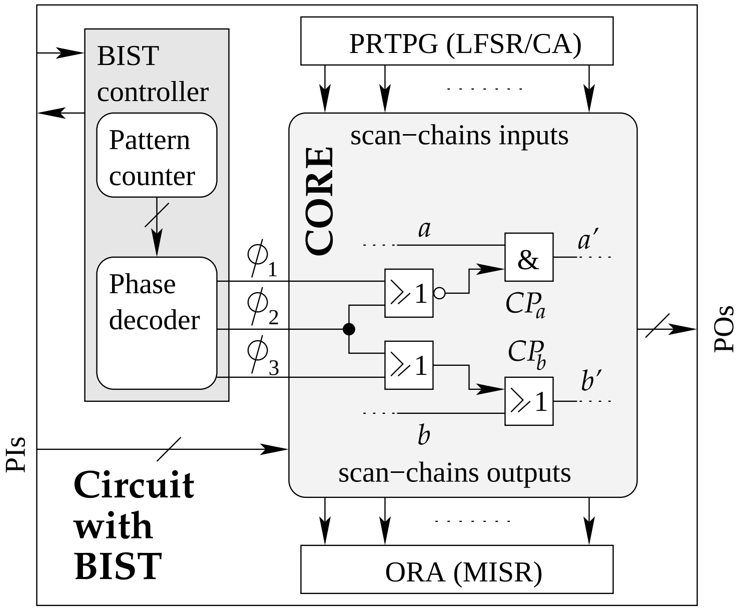
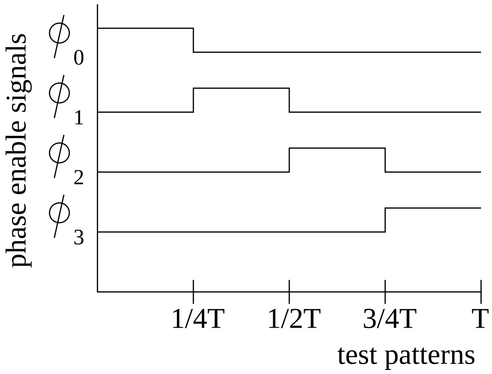


Figure 3.13: MTPI BIST scheme



	a'	b'
ϕ_0	a	b
ϕ_1	0	0
ϕ_2	0	0
ϕ_3	a	b

Figure 3.14: Impact MTPI phase decoder outputs on CPs

The MTPI BIST scheme will be described using the MTPI algorithm listed in Algorithm 3.4. Given as inputs the CUT, the test length (*number of PR patterns*), the target fault coverage, the maximum number of CPs and OPs (CP_{max}, OP_{max}), the number of phases (Φ) and the minimum-benefit-per-cost (*MBPC*) parameter, the objective of the algorithm is to perform circuit modification such that the target fault coverage is met within the specified test length, while satisfying the constraints on the number of CPs/OPs, with a minimum number of phases. The MBPC value is a parameter which can be used to tune the algorithm and will be described further on.

In each phase, a collection of CPs and OPs are inserted iteratively by assessing their

Algorithm 3.4 MTPI BIST

Input: CUT, number of PR patterns ($\boldsymbol{T}$), Target fault coverage ($\boldsymbol{FC}_{target}$), $\boldsymbol{CP}_{max}$, $\boldsymbol{OP}_{max}$, number of phases ($\boldsymbol{\Phi}$), Minimum-benefit-per-cost ($\boldsymbol{MBPC}$)

Output: CUT with TPs that reaches $\boldsymbol{FC}_{target}$

```
Perform fault simulation on $T$ patterns
if (fault coverage $>= FC_{target}$) then
    exit
for Phase 0 to $\Phi$-1 do
    Get the list of undetected faults

    { CPs selection }
    while (#$CPs < CP_{max}$) do
        Perform probabilistic fault simulation
        List the set of CP candidates
        Evaluate the set of CP candidates
        Select the best OP candidate: $CP_{best}$
        if ($CP_{best}$ meets $MBPC$ criterion) then
            Insert CP $CP_{best}$
        else
            break

    { OPs selection }
    Perform probabilistic fault simulation
    while (#$OPs < OP_{max}$) do
        List the set of OP candidates
        Evaluate the set of OP candidates
        Select the best OP candidate: $OP_{best}$
        if ($OP_{best}$ meets $MBPC$ criterion) then
            Insert OP $OP_{best}$
        else
            break

    Perform fault simulation on $T$ patterns
    if (fault coverage $>= FC_{target}$) then
        exit
```

impact on the fault coverage. The consequences of inserting a CP/OP at a circuit line are determined by means of a technique called *probabilistic fault simulation (PFS)* . The PFS technique [Tam96] computes detection probabilities for the set of undetected faults at different lines in the circuit, using analytical methods. This information, along with the detection threshold, is then used to guide the selection of TPs. The value of the detection threshold needs to be determined by considering factors like the phase duration (number of patterns in a phase) and the desired detection confidence. This detection threshold can be varied across different phases.

The design of a particular phase is concluded when it is determined that the expected benefit of a CP/OP does not justify the associated cost of implementation. The user-defined threshold *minimum-benefit-per-cost (MBPC)* is used for this purpose. The final step is to fault simulate the phase so that the list of undetected faults required for the design of a subsequent phase is determined.

OP Selection

The purpose of this step is to identify a pre-specified number of lines that enable the detection of a maximum number of faults. The objective is to select the lines such that the detection probability of a maximum number of faults meets a user specified threshold *DTh*. A three steps process is followed to achieve this objective.

First, probabilistic fault simulation is performed to determine the propagation profile. This information is represented as a sparse matrix $TM_{L\times F}$, with the L collected lines as rows, the F undetected faults as columns, and the probability of detecting a fault f on line l as entry $\mathrm{TM}_{l,f}$. In order to reduce the memory requirements, faults that propagate to a line with a probability less than a certain minimum threshold are dropped. In addition, lines that carry less than a minimum number of faults, as determined by the minimum-benefit-per-cost (*MBPC* (see Algorithm 3.4) criterion, are eliminated. The problem of selecting a pre-specified number of OPs, becomes equivalent to that of selecting the set of rows which maximizes the number of columns satisfying the detection probability *DTh*. This is an NP-complete problem, for which [Tam96] shows a greedy heuristic selection method to solve it.

CP Selection

The RPR problem cannot be entirely solved by inserting OPs alone, CPs are needed as well. In the MTPI scheme, CPs are driven by fixed values and are aimed at improving the excitation probability and propagation of specific faults. The insertion of CPs is divided into three steps:

1. Probabilistic fault simulation is performed to determine the propagation profile.
2. The set of candidate positions for CP insertion is determined.
3. The best CP candidate is selected, by determining the benefit of each candidate through incremental probabilistic fault simulation.

These steps are explained below:

The CP selection aims at quickly identifying the set of CP candidate lines. It is necessary to eliminate ineffective lines early on so that time spent in the subsequent selection step is reduced. However, good candidates must not be missed; they are determined by an estimation technique that computes two estimates, E_0 and E_1 for various lines in the circuit. These measures give an indication of the number of faults that could potentially be detected by placing an AND/OR-type CP respectively. Lines for which E_0 or E_1 exceeds a minimum acceptable value are retained for subsequent evaluation.

The computation of E_0 and E_1 utilizes the propagation profile information and is driven by three pre-specified parameters:

1. Minimum Probability Threshold *MPTh* which specifies the minimum acceptable value for 0(1) probability at a line in the circuit.

2. Low Threshold *LTh*

3. High Threshold *HTh*

A line for which the 0-probability(1-probability) falls below MPTh is called a 0-failing (1-failing) line. For each 0-failing (1-failing) line in the circuit, a constant 0-value (1-value) is injected at that line and the consequent signal probability changes are determined. The values LTh and HTh are used to record changes in signal probability of lines due to the insertion of a CP. A line for which the 0 or 1 probability changes from a value below LTh to a value above HTh after the insertion of a CP is said to facilitate the excitation and propagation of certain faults. For the purpose of estimation, such faults are considered to be detected. Hence, the estimation procedure predicts the number of faults that could be detected by a CP by reasoning about the impact of signal probability changes on the faults.

Starting from the CP candidate location, the probability changes are computed in a levelized, selective-trace manner. For each gate that is affected, new signal probabilities are obtained. Fan-outs of the affected gate are scheduled for evaluation only if the new signal probabilities deviate from the original signal probabilities by more than a pre-specified threshold. The signal probability changes, along with the threshold LTh and HTh, are then used to compute the estimates E_0 and E_1.

The rank of each CP candidate is defined as the number of additional faults that propagate to POs or OPs. The candidate with the highest rank is then selected for CP insertion.

Experimental results of the MTPI algorithm

Table 3.7 shows experimental results [Tam96], of the MTPI algorithm on the ISCAS'85 benchmark circuits c2670 and c7552 and five ISCAS'89 benchmark circuits. For each circuit first the redundant faults have been eliminated. The objective of the experiment is to achieve (near) complete fault coverage with 32,000 PR patterns, and a minimum of phases.

Before the TPI algorithm is run, the number of phases (Φ), the duration and the number of CPs and OPs, have to be determined. Φ is chosen to be an arbitrary value, and the entire test length among these phases is evenly distributed. Fault simulation for phase ϕ_0 is then performed using a fast fault simulator. The set of undetected faults is used to

Table 3.7: MTPI experimental results

Circuit	No TPI		Two-phase MTPI			Multi-phase MTPI			
	Average FC(%)	Max. FC(%)	#CP/ #OP	Average FC(%)	Max. FC(%)	Φ	#CP/ #OP	Average FC(%)	Max. FC(%)
c2670	88.12	88.82	1/5	100.0	**100.0**	-	-	-	-
c7552	96.73	97.06	18/2	99.39	99.60	6	18/2	100.0	**100.0**
s3330	87.95	88.99	1/4	99.97	**100.0**	-	-	-	-
s3384	96.06	96.30	0/5	100.0	**100.0**	-	-	-	-
s4863	97.66	98.12	6/4	99.78	99.83	4	6/4	99.96	**100.0**
s9234	93.13	93.79	8/10	99.80	99.85	3	8/10	99.97	**100.0**
s15850	95.72	95.99	15/17	99.16	99.21	4	15/17	99.67	99.71
s38417	95.40	95.64	18/30	99.79	99.82	3	18/30	99.81	99.85

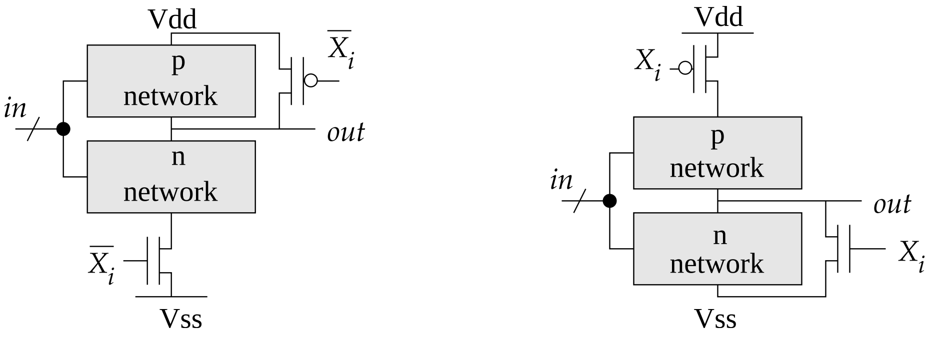
insert OPs. No CPs are inserted during phase ϕ_0. The design of each of the subsequent phases ϕ_1 to $\phi_{\Phi-1}$ is then carried out iteratively. In each phase a fixed number of CPs, targeting the set of undetected faults inherited from the previous phase, have been inserted with the knowledge of already inserted TPs. All OPs are inserted during phase ϕ_0 and the total number of CPs are distributed equally among the phases ϕ_1 to $\phi_{\Phi-1}$. The detection probability threshold DTh has been set to $\frac{4}{D_{\phi_i}}$, where D_{ϕ_i} is the test length of phase ϕ_i, and the MBPC parameter has been set to 10.

The Column *Circuit* in Table 3.7 shows the name of the circuit, Columns *No TPI*, *Two-phase MTPI* and *Multi-phase MTPI* TPI show experimental results for the circuits in case no TPs are inserted, respectively an MTPI run of two phases or an MTPI run with more than two phases took place. In order to reduce the random effect, the average fault coverage (*Average FC*) and maximum fault coverage (*Max. FC*) of ten fault simulation experiments are presented. In case of TPI, Column *#CP/#OP* shows the number of inserted CPs and OPs. For the multi-phase TPI runs, also the number of phases is shown in Column Φ.

The two-phase experiment results shows that all circuits achieve an average fault coverage greater than 99%, while without TPI, none of the circuits reach the 99% fault coverage level. Circuits c2670, s3330 and s3384 even reach complete (=100%) SAF fault coverage after TPI.

The goal of the experiment was to reach (near-) complete fault coverage for all listed circuits. For the circuits which did not result in complete fault coverage with the two-phase experiment, multi-phase TPI experiments (with $\Phi > 2$) are conducted. The results in Column *Multi-phase TPI* show that the number of phases that are necessary to achieve (near-) complete fault coverage is small, with a maximum of six phases for circuit c7552. This shows that only a few combinations of enabled/disabled CPs are necessary. For circuits s15850 and s38417 also no complete fault coverage has been achieved with the multi-phase experiment, at least not with the given number of inserted CPs and OPs.

The MTPI BIST architecture is commercially available and [Het99, Fei01, Gu 01]



(a) A 1-controllable gate (b) A 0-controllable gate

Figure 3.15: AND/OR TP alternatives

provide case studies of the MTPI algorithm on industrial designs. They show that MTPI BIST is applicable to industrial circuits as long as the circuits are BIST & TPI ready, or can be made BIST & TPI ready, i.e., meet the requirements for BIST & TPI implementation as given in Sections 2.3 and 3.4.

3.7.4 Other state-of-the-art TPI algorithm

In Schotten et al. [Sch95], a TPI algorithm is presented for an area efficient BIST. Like the adjusted CRF TPI algorithm presented in [Che95b], the TPI algorithm in [Sch95] does not require a full-scan circuit. Like the (H)CRF TPI algorithms, [Sch95] uses controllabilities, observabilities and a cost function to determine the best TP positions in the circuit, although different controllability, observability and cost equations are used than the COP equation on which the (H)CRF TPI algorithms are based. [Sch95] claims that their algorithm is inefficient in case of large circuits with poor controllability, therefore we have not looked further into this method.

In [Tou96] a test set dependent TPI method is presented, which is based on *path tracing*. Fault simulation is used to identify faults that are not detected by a given set of test patterns, e.g., a set of ATPG patterns. For each undetected fault, a path tracing procedure is used to identify the set of TPs that will enable the fault to be detected. This set is the *set of TP solutions* for the fault. Given the set of TP solutions for each undetected fault, a minimal set of TP is selected to achieve the desired fault-coverage, using a set covering procedure. The set covering procedure is an NP-complete problem. Although good heuristics exist [Chr75], this can result in very large CPU times for larger circuits with many undetected faults.

In [Sav91] and [You93] alternatives for AND/OR TPs are presented. Instead of AND and OR gate, the solution given in Fig. 3.15 is used. This is called the *Force-Observe*

(F-O) approach. When the $\overline{X_i}$ input of Fig. 3.15(a) is set to 0, the output (OUT) is forced to 1. Similarly, when the X_i input in Fig. 3.15(b) is set to 1, the output is forced to 0. Thus, when a value of a node is needed to control to one, the gate driving it is replaced by the controllable element of Fig. 3.15(a). If a node has to be controlled to zero, the gate driving it is replaced by the element of Fig. 3.15(b). Simulations [You93] have shown that the speed degradations observed with the F-O approach are generally smaller than those introduced by the classical AND/OR TPs. The F-O TPs are useful to achieve a good controllability at the cost of a minimal speed degradation. With highly loaded nodes however, classical AND/OR TPs offer a better performance and they should preferably be used.

3.8 Overview TPI topics in this dissertation

Subsection 3.8.1 provides an overview of the TPI topics that are addressed in the remaining part of the dissertation. In order to test and compare the TPI algorithms and techniques that are proposed in the following chapters, several TPI benchmark circuits have been used. These circuits are described in Subsection 3.8.2

3.8.1 TPI topics overview

In the following three chapters, the application of TPI on improving BIST fault coverage, respectively on facilitating stuck-at fault ATPG, and on facilitating gate-delay fault ATPG, are addressed. In all cases, the HCRF TPI algorithm is used as a base for further development, as will be explained in Chapter 4. COP, that is used in the CRF and HCRF TPI algorithms, can only cope with Boolean values. Therefore COP will be extended in Chapter 4 such that the HCRF TPI algorithm becomes applicable to industrial circuits, i.e., circuits containing three-state elements and unknown values. In Chapter 4 a new cost function will be proposed with which better PR fault coverage improvements can be achieved. Also a technique is proposed that reduces the number of TP candidates in the circuit, without impacting the quality of the TP selection.

In Chapter 5 it will be shown that the HCRF TPI algorithm is not only able to improve BIST fault coverages, but is also able to reduce stuck-at fault test set sized generated by a compact ATPG tool. The HCRF TPI algorithm is not aimed on reducing the compact ATPG test set sizes. Therefore, in Chapter 5, new cost functions and techniques are introduced that are aimed on reducing stuck-at fault ATPG test set sizes, such that better test set size reductions can be achieved.

In Chapter 6 it will be shown what the impact is of TPI for stuck-at fault ATPG on the ATPG test set sizes for gate-delay faults. It will be shown that TPI significantly reduces gate-delay fault test set sizes as well.

3.8.2 TPI benchmark circuits

In order to evaluate the various TPI methods and techniques, several TPI benchmark circuits have been used in this dissertation. These benchmark circuits consist of circuits that are commonly used in literature for benchmarking test related algorithms, i.e., the ISCAS circuits [Brg85, Brg89, Kis89] and circuit (cores) that were actually used in commercial industrial products.

The properties of the ISCAS benchmark circuits used throughout this dissertation are described in Appendix A and consist of the following three groups of circuits:

1. **ISCAS'85**
   Eight of the ISCAS'85 benchmark circuits [Brg85] have been used in this dissertation for benchmarking TPI, i.e., circuits c880, c1355, c1908, c2670, c3540, c5315, c6288 and c7552. These ISCAS'85 circuits are all Boolean combinational logic circuits not containing any memory elements. The number in the name gives an indication of the number of signal lines in the circuit.

2. **ISCAS'89**
   In 1989 a new set of ISCAS benchmark circuits has been introduced by Brglez [Brg89] that consists of Boolean sequential circuits, circuits containing memory elements, i.e., flipflops. Eleven of these circuits have been benchmarked throughout this dissertation, i.e., circuits s1196, s1423, s1488, s1492, s5378, s9234.1, s13207.1, s15850.1, s35932, s38417 and s38584.1.

3. **ISCAS'89a**
   Also in 1989 a set of addendum circuits [Kis89] has been added to the original set of ISCAS'89 circuits. Ten of these ISCAS'89 addendum circuits have been used for benchmarking, i.e., circuits s499a, s635a, s938a, s1269a, s1512a, s3271a, s3330a, s3384a, s4863a and s6669a.

The industrial designs used for benchmarking TPI are all Philips industrial circuits. Properties of these industrial circuits are found in Appendix B. These industrial benchmark circuits consist of the following groups of circuits:

1. **Philips Boolean designs**
   Fifteen Boolean Philips designs have been used for benchmarking TPI, i.e., circuits p5973, p7653, p13138, p14148, p27811, p31025, p34592, p36503, p43282, p43663, p72767, p73133, p73257, p75344 and p162057. Like for the ISCAS circuits, the number in the name is an indication of the number of signal lines in the circuit. All of these listed circuits are cores/parts of commercial industrial ICs. Although being industrial designs, these circuits do not contain three-state elements. Only circuit p162057 contains fixed unknown inputs.

2. **Philips three-state designs**
   Besides Boolean circuits, also fourteen industrial designs containing three-state

elements and/or fixed unknown inputs are used for benchmarking, i.e., circuits p32118, p37021, p66171, p71553, p93140, p104649, p114605, p137498, p481470, p596922, p598004, p705050, p824184 and p854266. Circuits p71553 and p93140 are designed only for test purposes, the remaining circuits are parts of commercial designs.

3.9 Summary and conclusions

The purpose of test point insertion (TPI) is to insert extra logic, i.e., test points (TPs), into the circuit to improve the testability of the circuit. TPs consist of control points (CPs) and observation points (OPs). CPs are inserted to improve the controllability of lines and OPs to improve the observability of lines. Traditionally, CPs consist of an extra input connected to an extra inserted AND/OR gate at a line in the circuit. With the extra input, the output of the AND(OR) gate can be forced to 0(1), bypassing the logic in the fan-in cone of the line where the CP is inserted. OPs consist of extra inserted outputs. In scan-designs, CPs and OPs are often controlled and observed by extra SFFs. These SFFs themselves can also be used as TPs, at least when they are transparent during normal application mode.

TPI can be used to facilitate testing in different ways, e.g., it can be used to improve PR fault coverage or to facilitate the generation of ATPG patterns. TPI algorithms can be categorized, depending on the method they use to determine the faults which testability should be improved, into: ATPG, fault simulation, and testability analysis based methods. TPI methods can also be divided into methods that do take into account the test set that will be applied to the circuit, i.e., test set *dependent* methods, and TPI methods that do not take this information in account, i.e., test set *independent* methods.

In case of industrial circuits, TPI algorithms should be able to cope with three-state circuits and avoid bus-conflicts. Also TPs at the critical paths should be avoided to prevent performance loss.

Several TPI algorithms, e.g.,the CRF TPI algorithm [Sei91], the Hybrid CRF (HCRF) TPI algorithm [Tsa97] and the Multi-phase TPI (MTPI) algorithm [Tam96] have been described which improve the fault coverages. Both the CRF and HCRF TPI algorithm use a cost function, based on COP testability analysis measures, to select the best positions in the circuit to insert TPs. Therefore first descriptions of COP, the cost function and cost gradients have been given. Because most TPI algorithms for BIST assume a STUMPS architecture, also a description of the STUMPS architecture has been given.

Several sets of circuits have been described, i.e., ISCAS and Philips benchmark circuits, that are used in this dissertation to benchmark TPI. In the following three chapters TPI methods are described to facilitate BIST, facilitate ATPG, respectively facilitate gate-delay fault ATPG.

CHAPTER 4

Test Point Insertion for BIST

Section 4.1 starts with a comparison of the CRF, HCRF and MTPI algorithms described in Chapter 3 and summarizes their advantages and disadvantages. Also will be explained why the HCRF TPI algorithm is chosen as a base for further TPI development. Section 4.2 describes how COP can be extended such that it is also applicable to industrial circuits. Section 4.3 describes our proposed TPI algorithm for BIST. It includes experimental results that show that our proposed TPI algorithm results in significant PR fault coverage improvement and is applicable to complex large industrial circuits containing three-state elements. Also included in this section is a technique to speed-up the TPI algorithm without impacting the quality of TPI. Section 4.4 summarizes and concludes this chapter.

4.1 Comparison of the CRF, HCRF and MTPI algorithm

This section starts with comparing experimental results of the CRF, HCRF and MTPI algorithms for BIST in Subsection 4.1.1, followed by a summary and conclusions in Subsection 4.1.2. One of the main drawbacks of most TPI algorithms is that they do not support industrial circuits, i.e., circuits that contain lines that can become Z or U. The implications of industrial circuits on TPI and BIST are described in Subsection 4.1.3.

4.1.1 Comparison of the TPI experimental results

Table 4.1, taken from [Tsa97], shows a comparison of the TPI algorithms described in Section 3.7. The circuits, listed in Column *Circuit* have first been optimized by removing redundant faults from the fault list. Experimental results are shown for the CRF algorithm, respectively Hybrid CRF algorithm and MTPI algorithm. For the Hybrid CRF and MTPI algorithms, the number of CPs and number of OPs (Column *#CP/#WP*) are listed. No detailed information of the number of CPs and OPs is given for the CRF algorithm, therefore only the number of inserted TPs (Column *#TP*) is listed. The fault coverages

Table 4.1: Comparison between the CRF, Hybrid CRF and MTPI TPI algorithms

Circuit	CRF		Hybrid CRF		MTPI				
	#TP	FC(%)	#CP/ #OP	FC(%)	#CP/ #OP	Φ	FC(%)	Φ	FC(%)
c2670	5	100.00	4/1	100.00	1/5	2	100.00	-	-
c7552	10	99.66	3/7	99.98	18/2	2	99.49	6	100.00
s3330	92	99.39	7/6	99.13	1/4	2	99.97	-	-
s3384	9	99.78	0/8	100.00	0/5	2	100.00	-	-
s4863	2	99.99	1/1	100.00	6/4	2	99.78	4	99.96
s9324	19	98.51	15/3	99.88	8/10	2	99.80	3	99.97
s15850	34	98.82	23/8	99.16	15/17	2	99.16	4	99.67
s38417	46	99.43	28/20	99.79	18/30	2	99.79	3	99.81

after TPI and applying 32,000 PR patterns are shown in the Columns *FC*. For MTPI the fault coverages are shown for a two two-phase MTPI run and, for circuits c7552, s4862, s9234, s1585, and s38417, also for a multi-phase (more than two phases) MTPI run.

The CRF algorithm inserts more TPs than the Hybrid CRF and MTPI algorithms, while this does not result in higher fault coverages. Only for circuit s3330 a higher fault coverage is achieved by the CRF algorithm than by the Hybrid CRF algorithm. But the CRF algorithm inserts ±7 times the number of TPs inserted by the Hybrid CRF algorithm (92 versus 13) resulting in a ±7 times higher silicon overhead.

The Hybrid CRF and 2-phase MTPI results are almost the same, except for circuits c7552, s3330 and s4863. The Hybrid CRF inserts fewer TPs, while achieving higher fault coverages for circuit c7552 and s4863. But for circuit s3330, 2-phase MTPI inserts fewer TPs while achieving a higher fault coverage. For the remaining circuits, the number of inserted test points and the achieved fault coverage are the same, although the ratio between CPs and OPs differs.

Increasing the number of phases, see Column *MTPI*, during MTPI, results in higher fault coverages. Compared to the Hybrid CRF algorithm, multi-phase MTPI is able to reach higher fault coverages for circuit c7552, s9234, s15850 and s38417. However, MTPI inserts ten TPs more in circuit c7552 than the Hybrid CRF algorithm and the difference in fault coverage for circuit s38417 is only 0.02%. Multi-phase MTPI is still not able to reach the same fault coverage for circuit s4863 as the Hybrid CRF algorithm. It is possible that circuit s3330 needs a number of CPs to be enabled at the same time to have effect, or, to avoid conflicting values of two CPs, needs two CPs not to be enabled at the same time. On the other hand, it is possible that the COP based CRF and HCRF algorithms result in better TPI for circuit s4863 than PFS.

4.1.2 Summary of the CRF, HCRF and MTPI algorithms

CRF TPI: For each possible TP position in the CUT, the Cost Reduction Factor (CRF) is calculated. The CRF is an estimate for the cost reduction achieved by the TP and is calculated using local testability analysis (TA) information of a signal line, namely the COP controllability, observability and gradient values, no TA measures of surrounding lines are taken into account. The gradients of a line reflect the impact of a controllability/observability change on that line on the entire circuit. The CRFs are not accurate enough for selecting a good TP position. For this reason, a set of TP candidates is selected for which the Actual Cost Reduction Factor (ACRF) is calculated. The ACRF is calculated by temporarily inserting the TP candidate in the CUT, recalculating the cost and taking the difference between the original cost and the cost after TPI. The TP that results in the largest ACRF will be implemented.

Advantage: The computation of the CRFs and gradient values are linear with the number of faults. They can be calculated fast.

Disadvantage: The CRFs are not accurate enough for TPI and calculating the ACRF for every TP candidate is too time-consuming. The ACRF is only calculated for a limited set of TP candidates; it is possible that good TPs are excluded because they are not included in the limited set of TP candidates.

Hybrid TPI: The Hybrid CRF algorithm uses the Hybrid Cost Reduction Factor (HCRF) as estimate for the ACRF. The HCRF is calculated by explicitly calculating the changes in detectability probabilities of a line when the changes in controllability or observability are too large, and using the gradients when the changes are small. The calculation of the HCRF happens in an event-driven procedure. Starting from the TP, the observability and controllability changes are propagated toward PIs and POs until the impact of the changes on the cost becomes below a given threshold. The HCRF values are more accurate than the CRFs and no ACRF calculation is needed to find a good TP position.

Advantage: The HCRFs are more accurate than the CRFs. *All* TP candidates are taken into account.

Disadvantage: Calculating an HCRF is slightly more time-consuming compared to the CRF. However, the CRF algorithm requires ACRF calculation which is not required for HCRFs. This algorithm is still based on TA estimates and does not take into account special information on the test set that will be applied to the circuit during test.

Multi-phase TPI: The COP based algorithms enable their CPs independently and equiprobably during the entire test. In the Multi-phase TPI (MTPI) algorithm, during any phase only one set of CPs is enabled and the other CPs are not. During the next phase, another set of CPs is enabled. The same CPs can be in multiple sets and so be enabled during multiple phases. This algorithm first uses fault simulation to find the controllabilities of all lines and to get a list of undetected faults. Probabilistic Fault Simulation (PFS) is used to determine the propagation profile of the faults. Estimates are used which reflect the number of undetected faults that will be detected by inserting a TP. These estimates are calculated with the data found with PFS. TPs which lead to the largest number of faults to become detected during a phase, are selected.

Advantage: CPs that have a conflict when they are enabled at the same time can be put in different phases such that they will never by enabled simultaneously. The results of (P)FS are more accurate than those of COP, because (P)FS takes advantage of the test set that is used during test.

Disadvantage: This algorithm is much slower compared to the COP algorithms, because fault simulation and PFS are both more CPU time-consuming than the controllability/observability and cost calculations of COP. The lack of randomness in enabling CPs sometimes has a negative impact on the fault coverage.

The CRF/Hybrid CRF algorithms are based on TA. This is faster and easier than using (probabilistic) fault simulation as MTPI does. TA does not only give estimates for the controllability and observability, but it also introduces a way to estimate the impact of a TP on the CUT. Fault simulation can result in better estimates for controllability and observability, because the information of the test set used for testing can be taken into account, but how to estimate the impact of a TP on the CUT is harder.

The Hybrid CRF algorithm outperforms the CRF algorithm; it is faster (taking into account that for the CRF, also the ACRF values have to be calculated) and does result in higher fault coverages. The MTPI and Hybrid CRF algorithm have comparable results. The structure of a CUT has a large influence on determining which TPI algorithm is better. There are circuits for which conflicting CPs results in reduced fault coverage improvement; these circuits benefit from the MTPI approach. For other circuits, the equiprobable CP enabling method used in the Hybrid CRF TPI algorithm outperforms the fixed enabling method used in the MTPI algorithm.

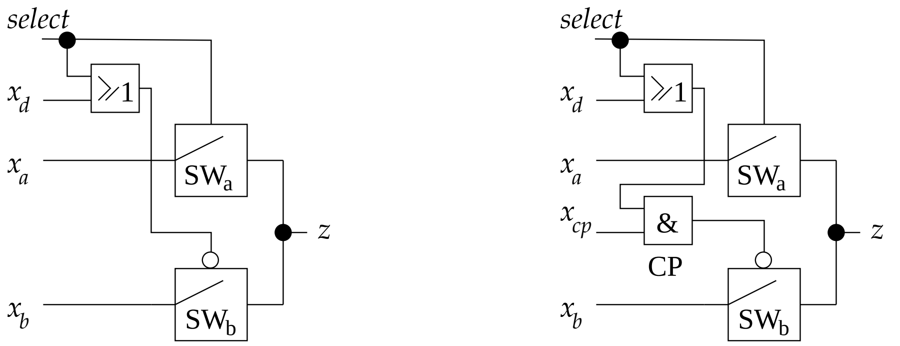
Case studies presented in [Het99, Fei01, Gu 01] have shown that the MTPI algorithm is also applicable to industrial designs. However, the MTPI TPI algorithm should be used in combination with the multi-phase BIST scheme depicted in Figure 3.13; it cannot be used in combination with other BIST models. As the MTPI algorithm is integrated in the MTPI BIST scheme, it is not applicable for other purposes than BIST, e.g., it is not applicable to facilitate ATPG. The CRF and HCRF TPI algorithms are not integrated in a BIST model and therefore are not limited to a single BIST model, or even to BIST. They can also be used to facilitate ATPG as will be shown in the following chapters. On the other hand, they assume to be applied to Boolean circuits only. They do not take into account any implications with respect to industrial circuits. The following subsection shows the problems that a TPI algorithm for BIST has to solve in order to be able to be applied to industrial circuits.

Because the HCRF TPI algorithm outperforms the CRF TPI algorithm, and the MTPI algorithm is only applicable to one single BIST scheme, i.e., the MTPI BIST scheme, the HCRF TPI algorithm is chosen as a base for further TPI development. The Hybrid CRF TPI algorithm is fast; the results given in Subsections 3.7.2 and 4.1.1, show that it can result in high fault coverages with only a few TPs; it is not very complex to implement, and because it is based on COP, it can be extended for industrial circuits as will be shown in the following sections. Although the COP controllability and observability estimates are not too accurate themselves, the event-driven mechanism of propagating the changes from the TP candidate, results in fast and good TP selection (see Tables 3.4 and 3.6). This event-driven mechanism can be seen as a basic method of determining the impact of TP candidates on a circuit. Even with TA measures other than COP, this event-driven mechanism with CFs is very useful as will be shown further on in this thesis. The algorithm is not limited to be used only for improving PR fault coverage in a BIST environment. For this reason, The Hybrid CRF algorithm is chosen as base for a new proposed TPI algorithm for industrial circuits, which can cope with large and complex industrial, including three-state elements.

4.1.3 Implications of Z and U values on TPI and BIST

Besides 0 and 1 values, also Z and U values occur in industrial designs. Floating values and unknown values can occur in case the circuit contains three-state elements, e.g., buses, switches, etc., or unknown fixed inputs, e.g., inputs from embedded memories. These Z and U values impact the TPI process as this does have an impact on the calculation of the COP TA measures and on PFS. Eq. 3.1 does not hold for lines that also have a probability on floating or unknown values. As a result, the COP equations given in Table 3.1 are not applicable to circuits that can contain these values. These circuits require extended COP equations which take into account probabilities on Z and U values. The same applies to PFS, also PFS has to take into account the probabilities on Z and U values in the circuit.

Another problem of TPI for BIST in industrial circuits, is the handling of buses. As described in Section 2.3, conflicting buses should be avoided during a BIST test run. Besides damaging the CUT, conflicting buses cause unknown values that could be captured by the MISR, making the complete MISR state unknown such that it cannot be checked whether the state of the MISR complies to a fault-free circuit. Inserting a TP nearby a bus, might enable a bus to come in a conflicting state during PR BIST. This is depicted in Fig. 4.1. The line *select* in Fig. 4.1(a) enables switch SW_a or switch SW_b (when $X_d = 0$) but never at the same time. However, due to the TP inserted before the control input of



(a) Circuit with non-conflicting bus (b) After TPI, a bus-conflict can occur

Figure 4.1: TPI on a circuit with buses can result in bus-conflicts

switch SW_b, a bus-conflict occurs when lines a and b drive opposite values, $select = 1$, $x_d = 0$ and $x_{cp} = 0$. When a TPI algorithm is used for facilitating ATPG, see Chapter 5, this possibility of a bus-conflict is not a problem. The ATPG algorithm has to avoid that the combination $select = 1$, $x_d = 0$ and $x_{cp} = 0$ occurs, which should not be a problem. However, in a PR BIST environment, it is not known in advance what the values will be for these signal lines, the combination $select = 1$, $x_d = 0$ and $x_{cp} = 0$ is possible, resulting in the nasty bus-conflict. Therefore a TPI for BIST algorithm should make sure that no TPs are inserted in lines that might result in extra bus-conflicts during PR test.

The TPI algorithm proposed in Section 4.3 is be able to cope with industrial circuits.

4.2 COP for industrial circuits

In order to be able to use COP, or the COP based HCRF TPI algorithm, with circuits that can contain Z or U values, we have extended COP to support probabilities on Z and U values. Subsection 4.2.1 shows how COP is extended with Z-controllability and U-controllability in order to deal with industrial circuits. Subsection 4.2.2 shows new observabilities for industrial circuits and Subsection 4.2.3 shows the COP detection probability equations for industrial circuits. The implications of the new COP TA measures for industrial circuits on the cost gradients are described in Subsection 4.2.4.

4.2.1 COP controllabilities in industrial circuits

In industrial circuits there are lines which do not only have a probability on being 0 or 1, but also on floating (Z) or being unknown (U). Therefore Eq. 3.1 does not hold for these circuits because it does not take into account these probabilities. Still the probabilities on all possible values for a signal line in an industrial circuit should add up to 1. Given this, Eq. 3.1 can been adjusted to Eq. 4.1, which does take into account float/unknown probabilities.

$$\begin{aligned} C0_l + C1_l + CZ_l + CU_l &= 1 \\ C0_l &= 1 - C1_l - CZ_l - CU_l \end{aligned} \tag{4.1}$$

In Eq. 4.1, CZ_l is the COP probability on floating for line l and CU_l is the COP probability on an unknown value. The controllability equations listed in Table 3.1 are based on Eq. 3.1 and are therefore not valid for industrial circuits. Still, the output of an OR gate is only 0 when all the input values are 0, hence the 0-controllability of OR gate output z ($C0_z$), can be found by multiplying the 0-controllabilities of all inputs ($C0_{x_i}$). In case of Boolean circuits, if the output is not 0, it is 1 and $C1_z$ of the OR gate is found by $1 - C0_z$. This is not valid in industrial circuits. However, it is still true that the output of an OR gate is only not 1 when *none* of the X inputs of the OR gate are 1. Hence $C1_z$ of the output of the OR gate is 1 minus the probability that none of the inputs are 1. The probability that an input x_i is not 1, is $1 - C1_{x_i}$. Now $C1_z$ can be calculated with Eq. 4.2.

$$C1_z = 1 - \prod_{i=1}^{X} (1 - C1_{x_i}) \tag{4.2}$$

The COP controllability equations of the other Boolean elements have been found in a similar way and are listed in Appendix C.

The outputs of Boolean gates cannot be at high impedance, therefore the Z-controllability of an output z (CZ_z) of a Boolean gate is always 0. Outputs of Boolean gates can be unknown if there are inputs with unknown or floating values, therefore CU_z of a Boolean gate can be non-zero. Because $CZ_z = 0$ for Boolean gates, CZ can be determined with Eq. 4.3.

$$CU_z = 1 - C1_z - C0_z \tag{4.3}$$

The controllabilities for the three-state elements can be determined in a similar way as for the Boolean elements. A three-state bus results in a 0 when one or more inputs are 0 and the remaining inputs are floating (Z). The probability that an input x_i is 0 or Z is found with $C0_{x_i} + CZ_{x_i}$. The probability that all lines are 0 or Z can be found by multiplying $(C0_{x_i} + CZ_{x_i})$ over all inputs. However, the bus only results in a 0, when *at least one* input is 0, therefore the probability that *all* inputs are Z should be excluded, hence the equation for calculating $C0_z$ for a three-state bus becomes:

$$C0_z = \prod_{i=1}^{X} (C0_{x_i} + CZ_{x_i}) - \prod_{i=1}^{X} (CZ_{x_i}) \tag{4.4}$$

$C1_z$ of the three-state bus is found in the same way and is given by:

$$C1_z = \prod_{i=1}^{X} (C1_{x_i} + CZ_{x_i}) - \prod_{i=1}^{X} (CZ_{x_i}) \tag{4.5}$$

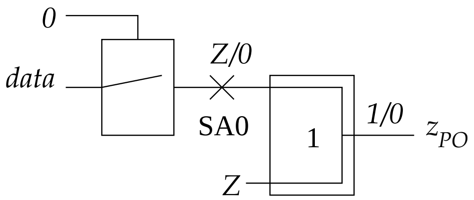


Figure 4.2: A Z↔0 discrepancy leading to a 0↔1 discrepancy on a PO.

The three-state bus only floats when all its input are floating, therefore CZ_z can be found by multiplying the probability that each input is floating or:

$$CZ_z = \prod_{i=1}^{X}(CZ_{x_i}) \tag{4.6}$$

The COP controllability equations for other three-state elements are found in a similar way and are listed in Appendix D.

4.2.2 COP observabilities in industrial circuits

In Subsection 3.5.1, the COP observability of a line l has been defined as the probability that a value change on line l will lead to a value change on at least one PO. In Boolean circuits, only the values 0 and 1 occur, therefore a value change on line l automatically means a 0↔1 change on l. But in industrial circuits, a value change does not automatically mean a 0↔1 change. A Z↔0 or a Z↔1 change on line l can also result in a 0↔1 change on a PO, only with other probabilities.[1] Fig. 4.2 shows an example of a Z↔0 value change which leads to a 0↔1 change on PO z_{PO}. Without the SA0 fault, the pull-up bus would pull the undriven bus to 1. In case there is a SA0 fault, the bus is driven to 0.

From the primitive three-state elements listed in Appendix D, only the wired AND, wired OR, pull-down bus and pull-up bus have the ability to convert a Z↔0 or a Z↔1 change into a Boolean 0↔1 change that can be detected on a PO.

A U value will neither lead to detectable value changes on gate outputs nor on POs (see the truth tables in Appendices C and D), therefore there exist no U↔0/1/Z observabilities. Because of the introduction of the Z value, not only the 0↔1 observability exists, but also the Z↔0 and Z↔1 observability:

W_l: The original COP observability, the probability that a 0↔1 change on line l results in a 0↔1 change on a PO.

WZ_l^0: The probability that a Z↔0 value change on line l results in a 0↔1 change on a PO.

---

[1] It is assumed that only 0↔1 value changes are detectably on a PO. Therefore the observabilities of all other value changes that can occur on a PO are 0.

WZ_l^1: The probability that a Z↔1 value change on line l results in a 0↔1 change on a PO.

Boolean gates cannot propagate Z values, a Z on an input will result in an U on the output, therefore the WZ^0 and WZ^1 observabilities for all Boolean gates are 0. The W-observability equations for the Boolean gates remain the same as given in Table 3.1 except for the (N)XOR gate. An input of the (N)XOR gate is only observable when the inputs are not Z or U and therefore the observability of an (N)XOR gate with inputs x_i, x_j and output z becomes:

$$W_{x_j} = W_z \cdot (C0_{x_i} + C1_{x_i}) \tag{4.7}$$

An overview of the COP observability equations for Boolean gates can be found in Appendix C.

Several of the three-state elements listed in Appendix D are able to propagate Z values from inputs to output. These elements have a non-zero WZ^0 and WZ^1. How the observabilities equations for three-state elements are determined, is shown for an X input wired OR gate. The observability equations for the other three-state elements have been extracted in a similar way and are given in Appendix D.

A 0 ↔ 1 change on an input x_j of a wired OR can only be observed on the output when the other inputs carry the value 0 or are floating. In this case, when x_j carries a 0, the output becomes 0, when x_j carries a 1, the output becomes 1. If one of the other inputs would be U, and x_j carries a 0, the output is also U, therefore the other inputs should only be 0 or Z. The W_{x_j} observability equation becomes:

$$W_{x_j} = W_z \cdot \prod_{i=1, i \neq j}^{X} (C0_{x_i} + CZ_{x_i}) \tag{4.8}$$

A Z↔0 change on input x_j of the wired OR results in a value change on the output when *all* other inputs are floating (see the truth-table for the wired OR given in Appendix D). In this case, the output will be 0 when x_j =0 and will be Z when x_j =Z. Therefore, the Z↔0 on input x_j is only observable when also a Z↔0 change on the bus output can be observed. The WZ_j^0 observability equation becomes:

$$WZ_{x_j}^0 = WZ_z^0 \cdot \prod_{i=1, i \neq j}^{X} (CZ_{x_i}) \tag{4.9}$$

A Z↔1 change on input x_j of the wired OR results in a value change on the output when *all* other inputs carry a value 0 or are floating. If *at least one* input carries a 0, the output of the wired OR will be 0 when x_j =Z and 1 when x_j =1. When all other inputs are floating, the output will be Z when x_j =Z and 1 when x_j =1. As a result the WZ_j^1 observability equation becomes:

$$WZ_{x_j}^1 = W_z \cdot \prod_{i=1, i \neq j}^{X} (C0_{x_i} + CZ_{x_i}) + (WZ_z^1 - W_z) \cdot \prod_{i=1, i \neq j}^{X} (CZ_{x_i}) \tag{4.10}$$

4.2.3 COP detection probabilities in industrial circuits

The possibility of floating lines in the circuit also impacts the COP detection probability equations. Not only can a SA0 fault on a line *l* be detected when a 0 is put on line and a 0↔1 change on *l* can be observed on a PO, but also when line *l* floats and a Z↔0 change on *l* can be observed on a PO. The COP detection probability estimates for the SA0 and SA1 faults on line *l* become:

$$\begin{aligned} Pd_{l/SA0} &= C1_l \cdot W_l + CZ_l \cdot WZ_l^0 && (4.11) \\ Pd_{l/SA1} &= C0_l \cdot W_l + CZ_l \cdot WZ_l^1 && (4.12) \end{aligned}$$

4.2.4 The cost gradients equations in industrial circuits

There are multiple types of COP controllabilities and COP observabilities in industrial circuits, and for each controllability/observability there exists a gradient with respect to the CF, namely dK/d$C0$, dK/d$C1$, dK/dCZ, dK/dW, dK/dWZ^0 and dK/dWZ^1.

Changing the observability of a PI x still only influences the COP detection probabilities of PI x and hence also only influences the cost contribution of the SA0 and SA1 on x. Therefore, Eq. 3.15 can still be used for industrial circuits to get the dK/dW_x gradient for a PI. However, the detection probabilities for lines in industrial circuits differ from the ones in Boolean circuits. Eq. 3.16 has to be changed into Eq. 4.13, given that Eq. 3.11 is used as cost contribution for a fault.

$$\frac{dK}{dW_x} = \frac{-C1_x}{(C1_x \cdot W_x + CZ_x \cdot WZ_x^0)^2} + \frac{-C0_x}{(C0_x \cdot W_x + CZ_x \cdot WZ_x^1)^2} \tag{4.13}$$

A change in Z↔0-observability on a PI only affects the term of the CF with respect to the SA0 fault on the PI. Given Eq. 3.11 as cost contribution for a fault, the dK/dWZ^0 gradient at PI x becomes:

$$\frac{dK}{dWZ_x^0} = \frac{dK_x}{dWZ_x^0} = \frac{d(\frac{1}{Pd_{x/0}})}{dWZ_x^0} = \frac{-CZ_x}{(C1_x \cdot W_x + CZ_x \cdot WZ_x^0)^2} \tag{4.14}$$

The dK/dWZ^1 gradient on PI x can be found in a similar way and becomes:

$$\frac{dK}{dWZ_x^1} = \frac{dK_x}{dWZ_x^1} = \frac{d(\frac{1}{Pd_{x/1}})}{dWZ_x^1} = \frac{-CZ_x}{(C0_x \cdot W_x + CZ_x \cdot WZ_x^1)^2} \tag{4.15}$$

In most cases, only zeros and ones are assigned to PIs. In that case, CZ_x will be 0 and all dK/dWZ_x^0 and dK/dWZ_x^1 gradients are 0.

When a gate is connected to PIs, changing the observability of the output z of the gate also results in observability changes on its inputs. Thus the CF will not only change by the terms with respect to the SA0 and SA1 fault on line z, but also by the observability changes on the inputs of the gate. Since it is known how the cost changes with respect to observability changes on the inputs (e.g., PIs), it is possible to apply a chain-rule to compute the dK/dW, dK/dWZ^0 and dK/dWZ^1 values of gate output z and the observability gradients for internal nodes in the CUT become:

$$\begin{aligned}\frac{dK}{dW_z} &= \frac{dK_z}{dW_z} \\ &+\sum_{i=1}^{X}\left(\frac{dK}{dW_{x_i}}\frac{dW_{x_i}}{dW_z}+\frac{dK}{dWZ^0_{x_i}}\frac{dWZ^0_{x_i}}{dW_z}+\frac{dK}{dWZ^1_{x_i}}\frac{dWZ^1_{x_i}}{dW_z}\right)\end{aligned} \tag{4.16}$$

$$\begin{aligned}\frac{dK}{dWZ^0_z} &= \frac{dK_z}{dWZ^0_z} \\ &+\sum_{i=1}^{X}\left(\frac{dK}{dW_{x_i}}\frac{dW_{x_i}}{dWZ^0_z}+\frac{dK}{dWZ^0_{x_i}}\frac{dWZ^0_{x_i}}{dWZ^0_z}+\frac{dK}{dWZ^1_{x_i}}\frac{dWZ^1_{x_i}}{dWZ^0_z}\right)\end{aligned} \tag{4.17}$$

$$\begin{aligned}\frac{dK}{dWZ^1_z} &= \frac{dK_z}{dWZ^1_z} \\ &+\sum_{i=1}^{X}\left(\frac{dK}{dW_{x_i}}\frac{dW_{x_i}}{dWZ^1_z}+\frac{dK}{dWZ^0_{x_i}}\frac{dWZ^0_{x_i}}{dWZ^1_z}+\frac{dK}{dWZ^1_{x_i}}\frac{dWZ^1_{x_i}}{dWZ^1_z}\right)\end{aligned} \tag{4.18}$$

A change in $C0$ on a PO z only affects the cost term with respect to the SA1 fault on PO z (it is assumed that $C1$ and CZ remain the same). Similarly, a change in $C1$ only affects the cost term with respect to the SA0 fault. On POs, a change in CZ has no influence on the SA0 or SA1 fault detectabilities and hence no influence on the cost contribution of the SA0 and SA1 faults on PO z. Because we assume that a Z value cannot be observed on a PO, Eqs. 4.11 and 4.12 on PO z reduce to:

$$Pd_{z/SA1} = C0_z \cdot W_z \tag{4.19}$$

$$Pd_{z/SA0} = C1_z \cdot W_z \tag{4.20}$$

Using these detection probability equations, the dK/d$C0$, dK/d$C1$ and dK/dCZ gradient values on PO z become:

$$\frac{dK}{dC0_z} = \frac{dK_{z/SA1}}{dC0_z} = \frac{d(\frac{1}{Pd_{z/SA1}})}{dC0_z} = \frac{-W_z}{(C0_z \cdot W_z)^2} \tag{4.21}$$

$$\frac{dK}{dC1_z} = \frac{dK_{z/SA0}}{dC1_z} = \frac{d(\frac{1}{Pd_{z/SA0}})}{dC1_z} = \frac{-W_z}{(C1_z \cdot W_z)^2} \tag{4.22}$$

$$\frac{dK}{dCZ_z} = 0 \tag{4.23}$$

A controllability change on input x_j of gate a does not only influence the detection probability of that input, but possibly also all the observabilities of the other inputs and the controllabilities on the output(s) of the gate and thus influences the detection probabilities ant the cost contribution of the faults on the inputs and inputs of the gate. Therefore the gradient equations become:

$$
\begin{aligned}
\frac{dK}{dC0_{x_j}} &= \frac{dK_{x_j/SA1}}{dC0_{x_j}} && (4.24)\\
&+ \sum_{i=1,i\neq j}^{X}\left(\frac{dK}{dW_{x_i}}\frac{dW_{x_i}}{dC0_{x_j}} + \frac{dK}{dWZ^0_{x_i}}\frac{dWZ^0_{x_i}}{dC0_{x_j}} + \frac{dK}{dWZ^1_{x_i}}\frac{dWZ^1_{x_i}}{dC0_{x_j}}\right)\\
&+ \sum_{k=1}^{Z}\left(\frac{dK}{dC0_{z_k}}\frac{dC0_{z_k}}{dC0_{x_j}} + \frac{dK}{dC1_{z_k}}\frac{dC1_{z_k}}{dC0_{x_j}} + \frac{dK}{dCZ_{z_k}}\frac{dCZ_{z_k}}{dC0_{x_j}}\right)\\
\frac{dK}{dC1_{x_j}} &= \frac{dK_{x_j/SA0}}{dC1_{x_j}} && (4.25)\\
&+ \sum_{i=1,i\neq j}^{X}\left(\frac{dK}{dW_{x_i}}\frac{dW_{x_i}}{dC1_{x_j}} + \frac{dK}{dWZ^0_{x_i}}\frac{dWZ^0_{x_i}}{dC1_{x_j}} + \frac{dK}{dWZ^1_{x_i}}\frac{dWZ^1_{x_i}}{dC1_{x_j}}\right)\\
&+ \sum_{k=1}^{Z}\left(\frac{dK}{dC0_{z_k}}\frac{dC0_{z_k}}{dC1_{x_j}} + \frac{dK}{dC1_{z_k}}\frac{dC1_{z_k}}{dC1_{x_j}} + \frac{dK}{dCZ_{z_k}}\frac{dCZ_{z_k}}{dC1_{x_j}}\right)\\
\frac{dK}{dCZ_{x_j}} &= \frac{dK_{x_j/SA0}}{dCZ_{x_j}} + \frac{dK_{x_j/SA1}}{dCZ_{x_j}} && (4.26)\\
&+ \sum_{i=1,i\neq j}^{X}\left(\frac{dK}{dW_{x_i}}\frac{dW_{x_i}}{dCZ_{x_j}} + \frac{dK}{dWZ^0_{x_i}}\frac{dWZ^0_{x_i}}{dCZ_{x_j}} + \frac{dK}{dWZ^1_{x_i}}\frac{dWZ^1_{x_i}}{dCZ_{x_j}}\right)\\
&+ \sum_{k=1}^{Z}\left(\frac{dK}{dC0_{z_k}}\frac{dC0_{z_k}}{dCZ_{x_j}} + \frac{dK}{dC1_{z_k}}\frac{dC1_{z_k}}{dCZ_{x_j}} + \frac{dK}{dCZ_{z_k}}\frac{dCZ_{z_k}}{dCZ_{x_j}}\right)
\end{aligned}
$$

The first part (the first line of each equation) represent the change with respect to the detection probabilities on line x_j. The second part of the equations (the second line) represents the change in the cost with respect to all possible observability changes on the other inputs due to the controllability change on line x_j. The third part (the last line) represents the change in the cost with respect to all possible controllability changes on the outputs due to the controllability change on line x_j.

4.3 Proposed TPI for BIST algorithm for industrial circuits

As described in Subsection 4.1.2, the HCRF TPI algorithm has been chosen as a base for further TPI development. In Chapter 3, it has been described that in the IC industry (extra) SFFs are used to control and observe the TPs. Using *transparent SFFs (TSFF)* as TPs removes the necessity of using extra AND/OR gates for CPs. TSFFs also have the

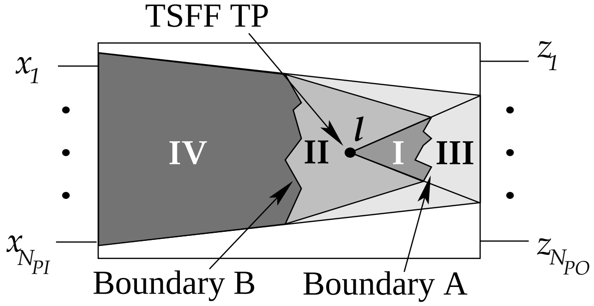


Figure 4.3: Computing HCRF for a TSFF TP

advantage over AND/OR CPs that they do not degrade the observability in the fan-in cone of the line at which the CP is inserted. For these reasons, the TSFF has been chosen as the TP type that will be inserted by the proposed Hybrid TPI based algorithms and proposed TPI techniques in the remaining part of this dissertation.

Subsection 4.3.1 describes the adjustments to the Hybrid CRF TPI algorithm in order to be able to apply the TPI algorithm to industrial circuits with TSFFs. Subsection 4.3.2 shows experimental results of the proposed TPI algorithm. Subsection 4.3.3 introduces a new CF that is used for determining the best TP positions. Subsection 4.3.4 introduces a technique that can be used to speed-up the TPI algorithm without impacting the quality of the TP selection.

4.3.1 The Hybrid CRF TPI algorithm for industrial circuits

The event driven mechanism of the Hybrid CRF TPI algorithm does not change for industrial circuits. However, the calculation of the Hybrid Cost Reduction Factor (HCRF) has to be adjusted due to the different CF and cost gradients, described in Subsection 4.2.4, and due to the usage of TSFFs.

Calculation of the HCRF for TSFF TPs in industrial circuits

Fig. 4.3 illustrates the calculation of the HCRF for a TSFF TP. Fig. 4.3 is similar to Fig. 3.11 for an AND/OR CP and also the calculation of the HCRF for a TSFF TP is quite similar to the HCRF for an AND/OR CP.

A TSFF TP changes both controllabilities (*C0*, *C1* and *CZ*) and observabilities (*W*, WZ^0 and WZ^1) in the circuit. The propagation of the altered COP values due to the TP has to proceed in both forward and backward directions. Starting from the TP *l*, the propagation of the new controllabilities calculation proceeds forward to the POs shown as Region I in Fig. 4.3. Because in industrial circuits also Z-controllabilities can occur, which influence the CF, not only the impact of C0 and C1 have to be taken into account, but also the impact of CZ on the cost. During the processing of each gate input x_{gate}, the ratio of dK/d$C0_{x_{gate}} \cdot \Delta C0_{x_{gate}}$ + dK/d$C1_{x_{gate}} \cdot \Delta C1_{x_{gate}}$ + dK/d$CZ_{x_{gate}} \cdot \Delta CZ_{x_{gate}}$ to $K^{(Org)}$ is compared with a given user-defined threshold to decide whether the controllabilities in its fan-out cone should be calculated or that the controllability gradients can be used to estimate the impact on the cost. $\Delta C0_{x_{gate}}$, $\Delta C1_{x_{gate}}$ and $\Delta CZ_{x_{gate}}$ represent the change in zero controllability, respectively one controllability and Z controllability, of input x_{gate} due to the TP.

After the forward propagation, a set of lines is obtained indicated as Boundary A in Fig. 4.3. These lines are used as starting points for the backward propagation of the observability changes to the PIs, similar to the calculation of the HCRF for OPs. This set of boundary lines can be seen as a set of pseudo-OPs. Line l is also added to this set of pseudo-OPs, because the observabilities on l are changed due to the inserted TSFF TP, which is both a CP and an OP.

In industrial circuits not only the 0↔1 observability changes (ΔW) impact the CF, but also the Z↔0 and Z↔1 observability changes (ΔWZ^0) and ΔWZ^1). Therefore the ratio of dK/d$W_x \cdot \Delta W_x$+dK/d$WZ_x^0 \cdot \Delta WZ_x^0$ + dK/d$WZ_x^1 \cdot \Delta WZ_x^1$ to $K^{(Org)}$ for each pseudo-OP is checked with a user-defined threshold to determine whether the explicit observability calculations (and hence detectability probability and cost calculations) have to be propagated backward any further. The backward calculation stops when this ratio becomes below this user-defined threshold. The set of lines where the backward calculation stops is indicated as Boundary B in Fig. 4.3 The observability gradients of the lines on Boundary B are used to estimate the cost impact on the remaining part of the circuit.

Given Eq. 3.11, the HCRF for a TSFF on line l comprises the following four parts:

1. $\sum_f \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)$ for every fault f inside Regions I and II.
   The new controllabilities ($C0$, $C1$ and CZ) and observabilities (W, WZ^0 and WZ^1) of the lines corresponding to the faults in Regions I and II and therefore the new detection probabilities and cost contributions of the faults are computed explicitly. They are computed using the above mentioned event-driven procedures.

2. $\sum_{l_{bA} \in Boundary\ A} \left(\frac{dK}{dC0_{l_{bA}}} \cdot \Delta C0_{l_{bA}} + \frac{dK}{dC1_{l_{bA}}} \cdot \Delta C1_{l_{bA}} + \frac{dK}{dCZ_{l_{bA}}} \cdot \Delta CZ_{l_{bA}} \right)$ for every fault in Region III.
   The controllability gradients of the lines l_{bA} on Boundary A are used to estimate $\sum_{f \in Region\ III} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)$.

3. $\sum_{l_{bB} \in Boundary\ B} \left(\frac{dK}{dW_{l_{bB}}} \cdot \Delta W_{l_{bB}} + \frac{dK}{dWZ^0_{l_{bB}}} \cdot \Delta WZ^0_{l_{bB}} + \frac{dK}{dWZ^1_{l_{bB}}} \cdot \Delta WZ^1_{l_{bB}} \right)$ for every fault in Region IV.
   The observability gradients of the lines l_{bB} on Boundary B are used to estimate $\sum_{f \in Region\ IV} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right)$.

4. The contribution of the new faults introduced by the inserted TSFF TP are added: $\frac{1}{Pd_{l'/SA0}}$ and $\frac{1}{Pd_{l'/SA1}}$. These are the SA0/SA1 faults on the TSFF output, see Fig. 3.4.d.

The equations to compute the HCRF for TSFF TPs become:

$$\begin{aligned} HCRF_l \quad = & \sum_{f \in Regions\ I\&II} \left(\frac{1}{Pd_f{}^{(Org)}} - \frac{1}{Pd_f{}^{(m)}} \right) - \left(\frac{1}{Pd_{l'/SA0}} + \frac{1}{Pd_{l'/SA1}} \right) \\ & - \sum_{l_{bA} \in Boundary\ A} \left(\frac{dK}{dC0_{l_{bA}}} \cdot \Delta C0_{l_{bA}} + \frac{dK}{dC1_{l_{bA}}} \cdot \Delta C1_{l_{bA}} + \frac{dK}{dCZ_{l_{bA}}} \cdot \Delta CZ_{l_{bA}} \right) \\ & - \sum_{l_{bB} \in Boundary\ B} \left(\frac{dK}{dW_{l_{bB}}} \cdot \Delta W_{l_{bB}} + \frac{dK}{dWZ^0_{l_{bB}}} \cdot \Delta WZ^0_{l_{bB}} + \frac{dK}{dWZ^1_{l_{bB}}} \cdot \Delta WZ^1_{l_{bB}} \right) \end{aligned} \tag{4.27}$$

The Hybrid CRF TPI algorithm for industrial circuits consists of the following seven steps:

Step 1: TP candidate lines determination
There are lines in the circuit which should be excluded as TP candidate, i.e. no TPs should be inserted at lines which can and should be able to float. TPs at such lines could result in bus-conflicts. Also neither TPs should be inserted at the control lines for bus-drivers nor at user-defined paths, e.g., critical paths or other “don’t touch” paths.

Step 2: Fault-set determination
Before TPI, the fault set that will define the CF and will impact the TPI process has to be determined; all untestable and collapsed faults are removed from the fault set.

Step 3: Determine COP TA values, cost and cost gradients
The COP TA measures and gradients for each line, and cost values for each fault in the circuit are calculated. Only faults which are in the fault set that contribute to the CF, as determined in Step 2, will get a non-zero cost contribution.

Step 4: Determine whether a TP should be inserted
The TPI process will only insert TPs when the cost is larger than a user-defined *desired maximum cost* and the number of already inserted TPs is smaller than the maximum number of TPs that are allowed to be inserted into the circuit.

Step 5: Calculate HCRF values
For each TP candidate line, the HCRFs for TSFF TPs are calculated.

Step 6: Insert a TP at the line with the highest HCRF value
Insert a TSFF at the line with the highest HCRF value when the highest HCRF value in the circuit is larger than the minimum required cost reduction for the insertion of a TP.

Step 7: Insert next TP or stop

If a TP has been inserted, go back to *Step 3* and insert more TPs, when necessary. If no TP has been inserted, the Hybrid CRF TPI for industrial circuits is finished, and the current circuit is returned.

In *Step 4,* the cost is compared with a user-defined *desired* cost. When the cost becomes below the desired cost, it is assumed that the fault coverage will be high enough. Instead of using this value, also a fault simulation run can be applied to the circuit with TPs to check whether the fault coverage is high enough. Although this is far more accurate than checking if the cost is below a desired cost, a fault simulation run can be very time consuming and therefore, massively repeating fault simulation is not always applicable for large industrial designs due to CPU time limitations.

4.3.2 Experimental results of the Hybrid CRF TPI algorithm for industrial circuits

Tables A.5 and A.6 in Appendix A show the fault coverage and fault efficiency of several ISCAS'85 respectively ISCAS'89 circuits after the application of 32,000 PR patterns. Table 4.2 shows the fault coverages and efficiencies for these circuits after the application of 32,000 PR patterns when TSFF TPs have been inserted with the proposed Hybrid CRF TPI algorithm for industrial circuits [Geu97b, Geu03]. TPs only have been inserted into the circuits that did not reach the 99% fault efficiency level without TPs. The experiments were performed on a AMD Athlon XP 1600+ machine, with 512MB DDR RAM memory running RedHat Linux 7.3.

Column *Circuit* lists the circuits name. Column *Without TPs* shows the number of applied PR patterns (*T*), the fault coverage (*FC(%)*) and the fault efficiency (*FE(%)*) for the listed circuits before TPI. Column *TPI* shows the number of inserted TSFF TPs (*TSFF*) in the circuit and the CPU time spent on TPI (*CPU*). Column *With TPs* shows the number of applied PR patterns (*T*), the fault coverage (*FC(%)*) and fault efficiency (*FE(%)*) for the circuit with TPs. The rows *Subtotal* show the subtotals for the ISCAS'85, respectively ISCAS'89 and ISCAS'89 addendum circuits. The fault coverages and efficiencies listed in these rows are weighted for the circuit size (ratio of the number of connections in the circuit to the total number of connections in all circuits). The row *Total* shows the overall results for the listed circuits.

Although the ISCAS benchmark do not contain three-state elements or fixed unknown input values, the Hybrid CRF TPI algorithm for industrial circuits is also applicable to these circuits. The TPI algorithm inserts TSFFs instead of AND/OR gates. Compared to the results given in Table 4.1, in general fewer TSFFs are inserted than CPs and OPs with the CRF TPI algorithm, the original Hybrid CRF TPI algorithm, or the MTPI algorithm, while reaching the same or higher fault efficiencies. The proposed Hybrid CRF algorithm for industrial circuits only has to insert one single TSFF TP in order to reach 100% fault efficiency for circuit c2670, while the other algorithms require at least

Table 4.2: PR Fault simulation results of ISCAS circuits using adjusted Hybrid CRF TPI with TSFF TPs

	Without TPs			TPI		With TPs		
Circuit	T	FC(%)	FE(%)	TSFF	CPU	T	FC(%)	FE(%)
c2670	32000	84.82	89.08	1	0.21 s	16064	95.74	100.00
c7552	32000	95.30	97.03	18	4.56 s	7872	99.12	100.00
Subtotal	64000	92.56	94.96	19	4.77 s	23936	98.24	100.00
s9234.1	32000	87.05	93.49	18	7.19 s	32000	94.03	99.76
s13207.1	32000	97.05	98.58	28	18.1 s	32000	98.80	99.99
s15850.1	32000	92.83	96.14	31	13.8 s	32000	97.68	99.89
s38417	32000	94.42	94.95	48	39.1 s	32000	99.51	99.98
Subtotal	128000	93.65	95.65	125	78.2 s	128000	98.35	99.94
s499a	32000	41.85	41.85	4	0.19 s	20608	100.00	100.00
s635a	32000	73.57	73.57	6	0.24 s	6304	100.00	100.00
s938a	32000	64.98	64.98	11	0.55 s	15136	100.00	100.00
s1512a	32000	95.28	95.28	4	0.24 s	6368	100.00	100.00
s3330a	32000	86.97	86.97	8	0.81 s	25408	100.00	100.00
s3384a	32000	96.18	96.18	5	0.26 s	5120	100.00	100.00
s4863a	32000	97.57	97.57	7	1.42 s	19488	100.00	100.00
Subtotal	224000	89.87	89.87	45	3.71 s	98432	100.00	100.00
Total	416000	92.99	94.72	189	86.6 s	250368	98.58	99.95

5 TPs. For all circuits, a fault efficiency of at least 99.7% has been reached with the proposed algorithm. 100% fault efficiency is achieved for all listed ISCAS'85 and ISCAS'89 addendum circuits and an average fault efficiency of 99.94% for the larger ISCAS'89 circuits. In total, an average fault efficiency of 99.95% has been reached.

After TPI, the circuits that reach 100% fault efficiency do not need to be tested with 32,000 PR patterns, but already reach this level with a much lower number of PR patterns. Circuit s3384 already reaches 100% fault efficiency after applying only 2624 PR patterns. The total insertion of 189 TPs in all circuits takes less than one and a half minutes (86.6 seconds).

Tables B.5 and B.6 in Appendix B show the fault coverages and efficiencies after the application of 32,000 PR patterns for respectively several Boolean, and three-state Philips industrial benchmark circuits. PR fault simulation has only been applied to the three-state circuits that do not suffer from possible bus-conflicts during the application of PR patterns. Table 4.3 shows the fault coverages and efficiencies for the industrial circuits after the insertion of TSFF TPs with the proposed Hybrid CRF TPI algorithm for three-state circuits [Geu97b, Geu03]. Again, TPs are only inserted into the circuits which do not reach 99% fault coverage without TPs. Except for circuit p73257, all Boolean industrial circuits, the first ten circuits in Table 4.3, reach 99% ore more fault efficiency

Table 4.3: PR Fault simulation results of industrial circuits using adjusted Hybrid CRF TPI with TSFF TPs

Circuit	Without TPs			TPI		With TPs		
	T	FC(%)	FE(%)	TSFF	CPU	T	FC(%)	FE(%)
p14148	32000	94.27	94.29	15	8.20 s	32000	99.26	99.27
p27811	32000	91.90	96.29	38	20.0 s	32000	95.32	99.70
p31025	32000	95.15	97.09	25	159 s	32000	98.17	99.41
p34592	32000	94.79	94.82	34	43.2 s	32000	99.89	99.92
p36503	32000	89.03	94.89	107	144 s	15488	94.22	100.00
p43282	32000	93.27	95.29	49	101 s	32000	97.51	99.52
p72767	32000	91.73	95.19	50	387 s	32000	96.58	99.39
p73133	32000	92.13	95.52	50	413 s	32000	96.41	99.39
p73257	32000	59.14	61.53	73	246 s	32000	78.09	80.44
p162057	32000	96.11	97.17	162	622 s	32000	98.74	99.80
Subtotal	320000	89.22	91.56	603	2146 s	303488	95.01	97.16
p32118	32000	84.07	90.61	32	25.3 s	32000	92.48	98.95
p37021	32000	67.95	91.78	37	28.3 s	32000	75.70	98.99
p114605	32000	89.22	90.47	80	415 s	32000	96.59	97.66
p137498	32000	93.79	95.49	137	323 s	32000	98.17	99.55
p481470	32000	82.82	83.73	185	2361 s	32000	91.72	92.57
p596922	32000	86.16	89.57	317	2886 s	32000	93.10	96.44
Subtotal	192000	85.48	88.30	788	6040 s	192000	92.94	95.64
Total	512000	86.56	89.24	1391	8186 s	495488	93.53	96.08

after TPI. Circuit p73257 seems to be a very RPR circuit. Although the fault efficiency increases with 19% (from 61% to 80%) after the insertion of 73 TPs, this is still far from the 99% level and far from an acceptable PR fault efficiency for the semiconductor industry. Overall, after the insertion of 603 TPs, the fault efficiency increases with 5.6%, from 91.5% to 97.1%. Inserting 603 TPs in the ten Boolean industrial circuits takes 2146 seconds, from which 622 seconds are spent on circuit p162057. The CPU time spent on TPI both increases with increasing number of TPs and increasing circuit size.

The last six circuits in Table 4.3 are three-state circuits that do not have conflicting buses with PR patterns. After TPI, still no buses should be able to become in conflict with PR patterns to avoid circuit damage and MISR problems within a BIST environment. Therefore, all connections that can be at high impedance state and all connections at which a TP can result in a bus-conflict are excluded as TP candidate. The PR fault coverages and efficiencies for the three-state circuits are lower than for the Boolean circuits. This is still the case after TPI. Only for circuit p137498 the 99% fault efficiency is reached (although circuits p32118 and p37021 have fault efficiencies very close to 99%). Still TPI results in significant increase in PR fault efficiency. The overall results of the three-state industrial circuits, show an increase of the fault efficiency from 88.3% to 95.6%. The overall results of all industrial benchmark circuits show an increase in fault efficiency of almost 7%, from 89.2% to 96.1%. 96% fault efficiency will results in a much lower DPM level than 89.2%, see Eq. 1.1 and is a far more acceptable fault coverage for the semiconductor industry than 89%.

4.3.3 New cost function for TPI for BIST

The CF used in the proposed Hybrid CRF algorithm has one disadvantage: it only focuses on improving the detection probability of the hardest-to-test faults, while also faults with a low detection probability, but not as low as the hardest-to-test faults, are ignored completely. For example, given a circuit with one fault f having a detection probability of 10^{-12} and ten other faults with a detection probability of 10^{-8}. All faults have a very low probability of being detected by a set of PR patterns. However the cost contribution of fault f is 10,000 times the cost contribution of the other ten faults. The TPI algorithm will only focus on improving the detection probability of fault f and ignores the other ten faults while their detection probability should also be increased.

In [Geu97b, Geu03] we proposed a new CF; it will take into account all faults with a low detection probability. With this CF even better fault coverages can be achieved using the same number of inserted TSFF TPs. This CF is based on the probability of a fault to be detected after *NPAT* independent PR patterns. The probability that a fault is not detected by a single PR pattern is:

$$P(f \text{ not detected}) = 1 - Pd_f \tag{4.28}$$

The probability that after *NPAT* independent PR patterns fault f is still not detected is:

$$P(f \text{ not detected after } NPAT \text{ patterns}) = (1 - Pd_f)^{NPAT} \tag{4.29}$$

This probability can also be used as a CF for a fault in the circuit. Faults with a low detection probability, i.e., that have a very low probability on being detected after *NPAT* patterns, will contribute to the cost with a contribution near 1, while faults with a relative high detection probability will have a cost contribution of almost 0. The cost contribution of a fault f becomes:

$$K_f = (1 - Pd_f)^{NPAT} \tag{4.30}$$

Hence the cost contribution of the SAF on line l becomes:

$$K_l = (1 - Pd_{l/SA0})^{NPAT} + (1 - Pd_{l/SA1})^{NPAT} \tag{4.31}$$

and finally the global CF for the circuit becomes:

$$\begin{aligned} K &= \sum_{f=1}^{F} \left((1 - Pd_f)^{NPAT} \right) && (4.32) \\ &= \sum_{l=1}^{L} \left((1 - Pd_{l/SA0})^{NPAT} + (1 - Pd_{l/SA1})^{NPAT} \right) && (4.33) \end{aligned}$$

A new CF results in new cost gradients. The chain-rules do not change, but the equations with respect to the impact of the controllability/observability change on the cost contribution of line *l* do. Given Eq. 4.31 the following equations are derived that should be used in the cost gradient Eqs. 4.16-4.17 and 4.24-4.26:

$$\begin{aligned}
\frac{dK_z}{dW_z} &= -NPAT \cdot C1_z \cdot (1-Pd_{z/SA0})^{NPAT-1} \qquad (4.34)\\
&\quad -NPAT \cdot C0_z \cdot (1-Pd_{z/SA1})^{NPAT-1}\\
\frac{dK_z}{dWZ_z^0} &= -NPAT \cdot CZ_z \cdot (1-Pd_{z/SA1})^{NPAT-1} \qquad (4.35)\\
\frac{dK_z}{dWZ_z^1} &= -NPAT \cdot CZ_z \cdot (1-Pd_{z/SA0})^{NPAT-1} \qquad (4.36)\\
\frac{dK_{x_j}}{dC0_{x_j}} &= -NPAT \cdot W_{x_j} \cdot (1-Pd_{x_j/SA1})^{NPAT-1} \qquad (4.37)\\
\frac{dK_{x_j}}{dC1_{x_j}} &= -NPAT \cdot W_{x_j} \cdot (1-Pd_{x_j/SA0})^{NPAT-1} \qquad (4.38)\\
\frac{dK_{x_j}}{dCZ_{x_j}} &= -NPAT \cdot WZ_{x_j}^0 \cdot (1-Pd_{x_j/SA0})^{NPAT-1} \qquad (4.39)\\
&\quad -NPAT \cdot WZ_{x_j}^1 \cdot (1-Pd_{x_j/SA1})^{NPAT-1}
\end{aligned}$$

As a result, also the HCRF equation changes and becomes:

$$\begin{aligned}
HCRF_l &= \sum_{f \in Regions\ I\&II} \left((1-Pd_f^{(Org)})^{NPAT} - (1-Pd_f{}^{(m)})^{NPAT} \right) \qquad (4.40)\\
&\quad - \left((1-Pd_{l'/SA0})^{NPAT} + (1-Pd_{l'/SA1})^{NPAT} \right)\\
&\quad - \sum_{l_{bA} \in Boundary\ A} \left(\frac{dK}{dC0_{l_{bA}}} \cdot \Delta C0_{l_{bA}} + \frac{dK}{dC1_{l_{bA}}} \cdot \Delta C1_{l_{bA}} + \frac{dK}{dCZ_{l_{bA}}} \cdot \Delta CZ_{l_{bA}} \right)\\
&\quad - \sum_{l_{bB} \in Boundary\ B} \left(\frac{dK}{dW_{l_{bB}}} \cdot \Delta W_{l_{bB}} + \frac{dK}{dWZ_{l_{bB}}^0} \cdot \Delta WZ_{l_{bB}}^0 + \frac{dK}{dWZ_{l_{bB}}^1} \cdot \Delta WZ_{l_{bB}}^1 \right)
\end{aligned}$$

Tables 4.4 and 4.5 show for several ISCAS, respectively industrial circuits, a comparison of the PR fault coverages achieved after TPI with the original CF of Tsai et al.[Tsa97] and TPI with the new *NPAT* CF, the *NPAT TPI algorithm* [Geu97b, Geu03]. The value of *NPAT* in CF 4.32 has been set to 32,000, the number of PR patterns that will be applied. In the Column *Original cost function* the number of applied PR patterns (*NPAT*), the PR fault coverages (*FC (%)*) and efficiencies (*FE (%)*) are shown in case the original CF is used. In the Column *TPI with NPAT cost function*, the number of inserted TPs (*TSFF*), the CPU time spent on TPI (*CPU*) and the number of applied PR patterns (*NPAT*), the PR fault coverages (*FC (%)*) and efficiencies (*FE (%)*) are shown for the *NPAT* TPI algorithm. The number of TPs inserted with the *NPAT* CF is the same as with the original CF.

The original CF already resulted in very good fault efficiencies for the ISCAS benchmark circuits as shown in Table 4.2. Except for the ISCAS'89 circuits, for all circuits

Table 4.4: Comparison of TPI with original and *NPAT* cost function for ISCAS circuits

	Original cost function			TPI with NPAT cost function				
Circuit	T	FC(%)	FE(%)	TSFF	CPU	T	FC(%)	FE(%)
c2670	16064	95.74	100.00	1	0.25 s	16064	95.74	100.00
c7552	7872	99.12	100.00	18	2.94 s	5280	99.17	100.00
Subtotal	23936	98.24	100.00	19	3.19 s	21344	98.27	100.00
s9234.1	32000	94.03	99.76	18	6.65 s	32000	94.08	99.83
s13207.1	32000	98.80	99.99	28	20.8 s	32000	98.76	99.98
s15850.1	32000	97.68	99.89	31	15.5 s	32000	97.72	99.92
s38417	32000	99.51	99.98	48	40.7 s	32000	99.51	99.98
Subtotal	128000	98.35	99.94	125	83.7 s	128000	98.36	99.95
s499a	20608	100.00	100.00	4	0.18 s	15104	100.00	100.00
s635a	6304	100.00	100.00	6	0.30 s	3456	100.00	100.00
s938a	15136	100.00	100.00	11	0.39 s	5440	100.00	100.00
s1512a	6368	100.00	100.00	4	0.29 s	24576	100.00	100.00
s3330a	25408	100.00	100.00	8	0.60 s	25408	100.00	100.00
s3384a	5120	100.00	100.00	5	0.31 s	2624	100.00	100.00
s4863a	19488	100.00	100.00	7	0.48 s	19488	100.00	100.00
Subtotal	98432	100.00	100.00	45	2.55 s	96096	100.00	100.00
Total	250368	98.58	99.95	189	89.4 s	245440	98.59	99.96

100% fault efficiency has been achieved. This does not change in case the *NPAT* CF is used. The fault coverage for circuit c7552 is higher with the *NPAT* CF than with the original CF while the fault efficiencies are 100% for both CFs. Hence the TPs inserted with *NPAT* TPI result in more untestable faults becoming testable after TPI than with the original CF for circuit c7552. Overall, the fault coverage and efficiency increases with 0.01% to 98.59% respectively 99.96% compared to the original circuit, while ±5000 fewer PR patterns need to by applied to the circuit. The CPU time spent on TPI has increased with 3 seconds from 86 seconds to 89 seconds. Overall, the differences between the two CFs are almost negligible for the ISCAS circuits.

The results for the industrial circuits given in Table 4.5 show more significant improvement with the *NPAT* TPI algorithm. For the Boolean industrial circuits, the fault efficiencies increase overall from 97.16% to 97.38% and the fault coverages from 95.01% to 95.22%. For 6 of the 11 Boolean circuits, the fault efficiencies are lower with the *NPAT* CF, but the differences are very very small, less than 0.1%. However the problem circuit within the Boolean industrial circuits, circuit p73257, clearly benefits from the *NPAT* CF. The fault efficiency increases with more than 2% which can be considered as a significant improvement. Also the three-state industrial circuits clearly profit from this new CF, the fault efficiency increment is 1.16%, from 95.64 to 96.80%. Especially the larger three-state circuits, p481470 and p596922, show significant improvement. These were the

Table 4.5: Comparison of TPI with original and *NPAT* cost function for industrial circuits

Circuit	Original cost function			TPI with NPAT cost function				
	T	FC(%)	FE(%)	TSFF	CPU	T	FC(%)	FE(%)
p14148	32000	99.26	99.27	15	9.08 s	32000	98.80	98.81
p27811	32000	95.32	99.70	38	21.0 s	32000	95.40	99.78
p31025	32000	98.17	99.41	25	197 s	32000	97.75	99.07
p34592	32000	99.89	99.92	34	28.7 s	32000	99.94	99.95
p36503	15488	94.22	100.00	107	135 s	32000	94.22	99.99
p43282	32000	97.51	99.52	49	114 s	32000	97.52	99.51
p72767	32000	96.58	99.39	50	448 s	32000	96.43	99.41
p73133	32000	96.41	99.39	50	472 s	32000	96.46	99.37
p73257	32000	78.09	80.44	73	159 s	32000	80.36	82.64
p162057	32000	98.74	99.80	162	865 s	32000	98.61	99.67
Subtotal	303488	95.01	97.16	603	2453 s	320000	95.22	97.38
p32118	32000	92.48	98.95	32	33.4 s	32000	92.10	98.59
p37021	32000	75.70	98.99	37	33.8 s	32000	75.48	98.76
p114605	32000	96.59	97.66	80	404 s	32000	96.92	98.01
p137498	32000	98.17	99.55	137	365 s	32000	98.21	99.63
p481470	32000	91.72	92.57	185	2265 s	32000	93.52	94.10
p596922	32000	93.10	96.44	317	3291 s	32000	94.55	97.87
Subtotal	192000	92.94	95.64	788	6394 s	192000	94.19	96.80
Total	495488	93.53	96.08	1391	8848 s	512000	94.49	96.97

circuits with lower fault efficiencies than the other three-state circuits, so it is more important that their fault efficiencies were improved in order to be able to reach a high enough test quality with PR testing. Overall the fault efficiencies have been increased with almost 1% from 96.08% to 96.97%. Especially the circuits with lower fault efficiencies ($<$ 98%) with the original CF benefit more from the new CF than the circuit that already reach a high fault coverage($>$ 98%) with the original CF.

These results show that the *NPAT* CF seems to be more useful for three-state circuits than the original CF. The *NPAT* CF for a fault, Eq. 4.30, is slightly more CPU time consuming than the original CF, Eq. 3.11. This can be seen in the CPU time spent on TPI. With the *NPAT* CF this increases from 8186 seconds to 8848 seconds.

4.3.4 CPU time reduction: Reduce the number of TP candidates

Not all lines in a circuit have to be considered as TP candidates. In Step 1 of the *NPAT* TPI algorithm, given in Subsection 4.3.1, already all PIs, POs, and internal three-state lines are excluded as TP candidate. However, there are more lines that can be excluded in order to reduce CPU time.

Table 4.6: Comparison results of TP candidates reduction for ISCAS circuits

		No candidates reduction			Candidates reduction		
Circuit	TSFF	FC(%)	FE(%)	CPU	FC(%)	FE(%)	CPU
c2670	1	95.74	100.00	0.25 s	95.74	100.00	0.22 s
c7552	18	99.17	100.00	2.94 s	99.17	100.00	2.38 s
Subtotal	19	98.27	100.00	3.19 s	98.27	100.00	2.60 s
s9234.1	18	94.08	99.83	6.65 s	94.03	99.77	4.21 s
s13207.1	28	98.76	99.98	20.8 s	98.76	99.98	12.4 s
s15850.1	31	97.72	99.92	15.5 s	97.73	99.94	10.2 s
s38417	48	99.51	99.98	40.7 s	99.51	99.98	30.9 s
Subtotal	125	98.36	99.95	83.7 s	98.35	99.95	57.7 s
s499a	4	100.00	100.00	0.18 s	100.00	100.00	0.15 s
s635a	6	100.00	100.00	0.30 s	100.00	100.00	0.19 s
s938a	11	100.00	100.00	0.39 s	100.00	100.00	0.32 s
s1512a	4	100.00	100.00	0.29 s	100.00	100.00	0.23 s
s3330a	8	100.00	100.00	0.60 s	100.00	100.00	0.52 s
s3384a	5	100.00	100.00	0.31 s	100.00	100.00	0.30 s
s4863a	7	100.00	100.00	0.48 s	100.00	100.00	0.46 s
Subtotal	45	100.00	100.00	2.55 s	100.00	100.00	2.17 s
Total	189	98.59	99.96	89.4 s	98.59	99.96	62.4 s

It is not necessary to take into account both the input and the output of an inverter/buffer as TP candidate. The observabilities, controllabilities and detectabilities are the same at the inverter/buffer output as at the inverter/buffer input (except for the $C0 \leftrightarrow C1$ and $\mathrm{Pd}_{SA0} \leftrightarrow \mathrm{Pd}_{SA1}$ swap in case of an inverter). When the output of the inverter/buffer is testable, the input is testable too. It is enough to only consider the output of an inverter/buffer as TP candidate.

Besides not taking into account inverter/buffer inputs as TP candidate, we have also experimented in [Geu02a] with not taking into account fan-out branches as TP candidate. Only the fan-out stem will be considered TP candidate. This results in a significant reduction of the number of TP candidates and hence in a reduction of the CPU time spent on TPI. However, excluding the fan-out branches can also impact the quality of TP selection because no TPs are taken into account for faults that benefit from changed controllabilities of a single fan-out branch due to a TP at that fan-out branch.

Tables 4.6 and 4.7 show for the ISCAS circuits, respectively industrial circuits, the impact of the TP candidates reduction on the fault coverages and CPU time spent on TPI [Geu02a]. Columns *Circuit* and *TSFF* show the name of the circuit, respectively the number of inserted TSFFs TPs. Columns *No candidates reduction* and *Candidates reduction* show the fault coverages (*FC(%)*), the fault efficiencies (*FE(%)*), and CPU time (*CPU*) spent on

Table 4.7: Comparison results of TP candidates reduction for industrial circuits

		No candidates reduction			Candidates reduction		
Circuit	TSFF	FC(%)	FE(%)	CPU	FC(%)	FE(%)	CPU
p14148	15	98.80	98.81	9.08 s	98.78	98.79	6.96 s
p27811	38	95.40	99.78	21.0 s	95.40	99.78	16.7 s
p31025	25	97.75	99.07	197 s	98.06	99.25	114 s
p34592	34	99.94	99.95	28.7 s	99.93	99.95	20.4 s
p36503	107	94.22	99.99	135 s	94.20	99.96	94.7 s
p43282	49	97.52	99.51	114 s	97.54	99.53	77.2 s
p72767	50	96.43	99.41	448 s	96.49	99.46	280 s
p73133	50	96.46	99.37	472 s	96.47	99.38	295 s
p73257	73	80.36	82.64	159 s	80.38	82.66	119 s
p162057	162	98.61	99.67	865 s	98.61	99.68	634 s
Subtotal	603	95.22	97.38	2453 s	95.25	97.40	1660 s
p32118	32	92.10	98.59	33.4 s	92.16	98.64	23.7 s
p37021	37	75.48	98.76	33.8 s	75.45	98.74	25.8 s
p114605	80	96.92	98.01	404 s	97.06	98.15	278 s
p137498	137	98.21	99.63	365 s	98.25	99.63	302 s
p481470	185	93.52	94.10	2265 s	93.58	94.16	1695 s
p596922	317	94.55	97.87	3291 s	94.56	97.89	2579 s
Subtotal	788	94.19	96.80	6394 s	94.23	96.84	4905 s
Total	1391	94.49	96.97	8848 s	94.52	97.00	6566 s

TPI in case no TP candidates reduction, respectively TP candidates reduction takes place.

The results in Table 4.6 show for the ISCAS circuits that there is almost no impact on the fault coverage when TP candidates reduction has been used. Only for circuit s9234.1, the fault coverage and fault efficiency is 0.05%, respectively 0.06% less than without TP candidates reduction. On the other hand, for circuit s15850.1 the fault coverage and fault efficiency slightly increase with 0.01%, respectively 0.02%. For the other circuits, there is no visible impact on the fault coverage.

Although the impact on the fault coverage is hardly visible, the impact on the CPU time spent on TPI is significant. The CPU time spent on TPI for the ISCAS circuits has been reduced from 89.4s to 62.4s; which means a 30% reduction. The CPU time reduction is less for the smaller ISCAS circuits, i.e., the ISCAS'85 (19%) and ISCAS'89 addendum (15%) circuits, than for the larger ISCAS circuits, i.e., ISCAS'89 circuits (31%). Especially for circuit s15850.1 a large reduction of 40% has been achieved.

The results of the industrial circuits given in Table 4.7 also show that there is not much impact on the fault coverage with TP candidates reduction. Overall, the fault coverage and fault efficiency do not reduce; they even increase with TP candidates reduction. Overall, the fault coverage and efficiency increase with 0.03%, the fault coverage from 94.49% to 94.52% and the fault efficiency 96.97% to 97.00%. Again, although the impact of TP candidates reduction is small on the PR fault coverage for the industrial circuits, the impact on the CPU time spent on TPI is much more significant. The CPU time has been reduced with 26% from 8848s to 6566s.

Hence the results of both the ISCAS and industrial circuits show that TP candidates reduction has no negative impact on the quality of the TP selection, there is no structural fault coverage reduction when TP candidates reduction is used. TP candidates reduction can impact the fault coverage, both positive (improve) as negative(reduction) but the difference is very small (in almost all cases less than 0.1%). On the other hand, the reduction in CPU time spent on TPI is substantial. CPU time reductions of ±30% can be achieved without impacting the quality of the TP selection.

4.4 Summary and conclusions

The MTPI BIST algorithm is commercially available and case studies have shown that the MTPI algorithm is applicable to industrial designs. But the MTPI algorithm has been designed to be used within the MTPI BIST scheme, depicted in Figure 3.13. Therefore it cannot be used in combination with other BIST models and is only applicable for PR fault coverage for BIST. The CRF and HCRF TPI algorithm are not restricted to a single BIST model or even to BIST, but they assume Boolean circuits and do not take into account any implications with respect to Z and U values in industrial circuits. Because the HCRF TPI algorithm is fast, outperforms the CRF TPI algorithm, and results in good PR fault coverage improvements after TPI, it has been chosen as a base for a new TPI algorithm that can cope with industrial circuits and results in even better PR fault coverage improvement after TPI. Extensions for COP TA measures, the CF and cost gradients have been proposed such that the improved HCRF TPI algorithm can cope with industrial circuits, containing possible Z and U values. SFFs are often used to drive CPs and observe OP outputs in (full-)scan circuits. Therefore the improved HCRF TPI algorithm inserts TSFFs such that the necessity of extra AND/OR gates for CPs is removed.

Experimental results of the HCRF TPI algorithm for industrial circuits, show that good PR fault coverage improvements can be achieved on both Boolean and three-state circuits and is applicable to both small and large complex industrial designs.

A new CF has been proposed for the TPI algorithm. The proposed improved Hybrid CRF algorithm for industrial circuits with the new CF, called *NPAT* TPI algorithm results in even better PR fault coverage than the original CF of Tsai, especially for the industrial three-state benchmark circuits.

In order to reduce the CPU time spent on *NPAT* TPI, a method has been introduced that reduces the number of TP candidates in the circuits without impacting the quality of the TP selection. Experimental results show that with this technique a 30% reduction in CPU time can be achieved, while keeping the same PR fault coverage improvement (after TPI) as without TP candidates reduction.

CHAPTER 5

Test Point Insertion for compact SAF ATPG

The increasing complexity of integrated circuits has a major impact on testing. The demands on the ATE become higher and higher, due to higher clock-frequencies and pin counts. This increasing complexity also results in larger ATPG test sets, resulting in higher ATE vector memory requirements and longer test application times; both resulting in increasing test and ATE costs. In order to cope with the increasing complexity of circuits, faster testers with much more vector memory will be required, which will become very expensive. In the international technology road-map from the SIA [Sem01], it is even expected that without the necessary solutions, the ATE will not be able to cope with the demands within only a few years.

An alternative to off-chip testing, using ATE, is on-chip testing; i.e., BIST, as described in Chapter 2. In many BIST implementations, a PRTPG is used to generate a large number of PR patterns that will be applied to the circuit. But such PRTPGs fail to detect the RPR faults, resulting in lower achievable fault coverages; reducing the quality of the test. In Chapter 4 it has been shown that the RPR problem can be solved by the insertion of TPs. The inserted TPs improve the PR testability of the RPR faults within the circuit.

TPI results in a reduction of the number of circuit inputs that have to be assigned in order to test the (RPR) faults with PR patterns. But the same holds for ATPG; after the insertion of a TP, also ATPG may have to assign fewer circuit inputs to find a pattern for the (RPR) faults. Because fewer assigned inputs are required to detect faults in the circuit after TPI, it becomes easier to combine patterns for different faults into one compact test pattern. When more tests for faults can be combined into one test pattern, overall, a more compact test set can be obtained.

In this chapter it will be shown that TPI, even if only targeted at solving the RPR problem (TPI for BIST), indeed results in more compact test sets for ATPG. However, sometimes a circuit is easily PR testable, while ATPG still does not result in a compact test set. Therefore also new TPI techniques will be described that lead to even smaller test sets than when using TPI for BIST for compact ATPG, and moreover, more consistently results in a significant reduction of test set sizes.

Section 5.1 shows the impact of TPI for BIST on ATPG test set sizes. Section 5.2 shows other TA measures that indicate if there are ATPG specific testability problems with respect to compact ATPG test sets in a circuit. Section 5.3 introduces the CF we proposed in [Geu00] that takes into account these new TA measures in order to get even better test set size reduction than with TPI for BIST. Although better test set size reduction has been achieved with the CF proposed in Section 5.3, several circuits still suffer from large test sets. They suffer from large *fan-out free regions (FFRs)*. In Section 5.4, four techniques are described to reduce the sizes of large FFRs and hence, the ATPG test set sizes. Section 5.5 introduces a TPI pre-process that analyzes the TA measures of a circuit to find its specific testability problems. Given the results of the analysis, a CF will be selected that is targeted at solving the hardest testability problem within that circuit. Experimental results in Section 5.5 will show that with the TPI pre-process, better ATPG test set sizes can be achieved. Section 5.6 summarizes and concludes this chapter.

5.1 Impact of TPI for BIST on ATPG test set sizes

This section shows that even TPI for BIST, which is only targeted at improving the PR fault coverage of a circuit, can already result in significant ATPG test set size reduction.

Tables A.3 and A.4 in Appendix A show ATPG test set size information for the ISCAS'85, respectively ISCAS'89, circuits without TPs. Tables B.3 and B.4 in Appendix B show ATPG test set size information for the Boolean industrial, respectively three-state industrial, benchmark circuits without TPs. The listed ATPG results in Appendices A and B are *compact* ATPG results. This means that the ATPG already applies ATPG techniques [Kon96a] to get a very small ATPG test set. In the remainder of this dissertation, with ATPG is meant compact ATPG. TPI will only be performed on circuits that have an ATPG test set size of more than 100 patterns. Circuits with smaller ATPG test set sizes are not candidate for TPI, because we consider their test set size to be small enough.

Tables 5.1 and 5.2 show experimental results of TPI for BIST (*NPAT* TPI) [Geu00] on compact ATPG test set sizes and ATPG generation times for several ISCAS and industrial benchmark circuits. In case of TPI for PR BIST, the number of PR patterns that will be applied during test is normally known and this value is used in the *NPAT* TPI algorithm as value for *NPAT*. In case of TPI for ATPG, the number of ATPG patterns in the test after TPI is not known in advance, so what value of *NPAT* should be used? The goal is to get a very small test set. Therefore we have set the value of *NPAT* to 256, to signal the *NPAT* TPI algorithm that the used test set is small and that the probability that faults are detected pseudo-randomly is low, because of the small test set.

Column *ATPG without TPs* shows the ATPG results for the circuits listed in Column *Circuit* without TPs; the ATPG test set size (*T*), the ATPG fault coverage (*FC(%)*) and the CPU time spent on ATPG (*CPU*) are listed. Column *TPI* shows the number of inserted

Table 5.1: Compact ATPG results of ISCAS circuits using *NPAT* TPI

Circuit	ATPG without TPs			TPI		ATPG after TPI		
	T	FC(%)	CPU	TSFF	CPU	T	FC(%)	CPU
c1908	114	99.71	1.32 s	25	1.03 s	36	99.92	0.46 s
c3540	119	96.38	2.59 s	25	3.00 s	75	97.80	1.74 s
c7552	128	98.55	6.01 s	40	6.06 s	60	99.47	3.16 s
Subtotal	361	98.13	9.92 s	90	10.1 s	171	99.08	5.36 s
s1196	133	100.00	0.52 s	20	0.58 s	45	100.00	0.30 s
s1488	115	100.00	0.53 s	20	0.92 s	90	100.00	0.48 s
s1494	117	99.46	0.52 s	20	0.91 s	89	99.87	0.49 s
s5378	117	98.87	2.81 s	30	2.17 s	57	98.93	1.56 s
s9234.1	135	93.95	20.0 s	50	6.77 s	53	94.85	4.46 s
s13207.1	272	98.87	15.5 s	50	7.72 s	223	99.22	11.9 s
s15850.1	128	97.51	13.9 s	50	9.13 s	98	98.30	10.1 s
s38584.1	123	95.57	44.9 s	50	22.4 s	122	97.33	39.7 s
Subtotal	1140	96.66	98.8 s	290	50.6 s	777	97.76	68.9 s
s499a	104	100.00	0.28 s	8	0.16 s	31	100.00	0.19 s
s938a	145	100.00	0.45 s	12	0.42 s	40	100.00	0.25 s
s3330a	163	100.00	2.20 s	32	1.65 s	61	100.00	0.99 s
Subtotal	412	100.00	2.93 s	52	2.23 s	132	100.00	1.42 s
Total	1913	97.00	111 s	432	63.0 s	1080	98.03	75.7 s

TSFF TPs (*TSFF*) and the CPU time spent on TPI (*CPU*). Column *ATPG after TPI* shows the ATPG results for the circuit after *NPAT* TPI; the ATPG test set size (*T*), the ATPG fault coverage (*FC(%)*) and the CPU time spent on ATPG (*CPU*) are listed for the circuits after TPI.

The experimental results given in Table 5.1 show that *NPAT* TPI is already capable of reducing the compact ATPG test set sizes of the ISCAS circuits from 1913 patterns down to 1080 patterns; a 43.5% ($\frac{1913-1080}{1913} \cdot 100\% = 43.5\%$) reduction. The variation in test set size reduction is large between the various circuits. For circuits c1908, s1196, s5378, s9234.1, and the ISCAS'89 addendum circuits reductions of over 40% are achieved (for circuit c1908 the test set size is reduced from 114 to 36 patterns; i.e., a 68% reduction), while for circuits s13207.1, and s38584.1 there are only reductions of 18% (from 272 to 223) and 0.8% (from 123 to 122).

These results show that the *NPAT* TPI algorithm is capable of reducing the compact ATPG test set sizes for most circuits, however the results for circuits s13207.1 and especially s38584.1 indicate that these circuits suffer from other testability problems than only poor PR random testability, because TPI for improving PR fault coverage is not enough to reduce the test set size for these circuits significantly. In Section 5.2 several other testa-

Table 5.2: Compact ATPG results of industrial circuits using *NPAT* TPI

	ATPG without TPs			TPI		ATPG after TPI		
Circuit	T	FC(%)	CPU	TSFF	CPU	T	FC(%)	CPU
p7653	334	99.48	14.9 s	15	2.85 s	153	99.63	8.25 s
p14148	385	99.99	30.0 s	15	3.48 s	218	100.00	17.4 s
p27811	200	95.21	42.8 s	38	12.8 s	168	95.29	32.7 s
p31025	714	97.90	307 s	25	16.6 s	449	98.67	129 s
p34592	243	99.88	142 s	34	17.0 s	162	99.98	56.1 s
p36503	175	97.20	71.3 s	30	14.4 s	110	97.22	37.1 s
p43282	382	97.66	207 s	49	28.0 s	207	97.70	123 s
p43663	194	99.23	58.8 s	50	21.9 s	98	99.25	36.0 s
p72767	438	96.31	1527 s	50	57.7 s	311	96.65	409 s
p73133	407	96.37	1492 s	50	59.2 s	301	96.73	405 s
p73257	2097	98.06	1030 s	73	71.8 s	2132	98.06	1030 s
p75344	273	99.18	153 s	55	45.9 s	196	99.25	114 s
p162057	2055	98.51	926 s	162	290 s	556	98.78	330 s
Subtotal	7897	97.95	6005 s	646	641 s	5061	98.15	2731 s
p32118	570	93.50	131 s	32	13.5 s	263	93.53	39.8 s
p37021	531	76.89	56.1 s	37	16.4 s	136	77.28	21.3 s
p66171	2042	96.79	576 s	66	102 s	1792	96.79	580 s
p71553	138	96.69	295 s	71	69.6 s	76	96.61	395 s
p93140	511	93.81	1139 s	93	107 s	367	94.25	616 s
p104649	1031	98.96	847 s	104	126 s	785	99.37	394 s
p114605	689	98.44	827 s	80	127 s	513	98.64	520 s
p137498	446	98.66	308 s	137	205 s	276	98.82	199 s
p481470	1679	99.23	16560 s	185	942 s	1671	99.25	12400 s
p596922	2038	96.87	8887 s	317	1895 s	2100	96.89	9475 s
p598004	2129	98.70	5051 s	100	633 s	1942	98.77	3006 s
p705050	655	76.76	8463 s	250	1749 s	530	91.47	4942 s
p824184	2066	96.54	130153 s	300	2470 s	1114	96.73	24468 s
p854266	565	97.54	6881 s	300	2613 s	359	97.62	5741 s
Subtotal	15090	94.29	180180 s	2072	11073 s	11924	96.58	62803 s
Total	22987	94.76	186185 s	2718	11715 s	16985	96.78	65535 s

bility problems that cause large test sets will be described, including a description of how TPI can help reducing these testability problems.

The overall ATPG fault coverage for the ISCAS circuits has been increased from 97% without TPs, to 98% with TPs. This can be considered as a significant improvement; 33% of the faults that were first not detected by the ATPG patterns become detected after TPI ($(1-\frac{(100\%-98\%)}{(100\%-97\%)})\cdot 100\% = 33\%$). Also after TPI, the CPU time spent on ATPG has decreased significantly; in total over all ISCAS circuits: from 111 seconds circuits without TPs to 75.7 seconds with TPs. This means a reduction of 31.8% ($\frac{(111-75.7)}{111} \cdot 100\% = 31.8\%$). Hence, TPI for BIST on the ISCAS circuits results in a significant ATPG test set size reduction, fault coverage improvement and CPU time reduction.

The experimental results of Table 5.2 also show that significant test set size reductions can been achieved with *NPAT* TPI for the industrial benchmark circuits. The total number of compact ATPG test patterns for the Boolean industrial circuits is reduced from 7897 patterns to 5061, a 35.9% reduction ($\frac{7898-5061}{7898} \cdot 100\% = 35.9\%$). The total number of compact ATPG test patterns for the three-state circuits is reduced from 15090 to 11924 patterns, or 21.0% reduction. The reduction for circuit p598004 is relatively small, only 187 patterns or 8.7%, while the number of patterns for circuit p596992 even increase. *NPAT* TPI for ATPG test set size reduction does not work very well on these circuits. The reason for this is explained in the following sections. In total over all industrial benchmark circuits, the number of ATPG patterns is reduced from 22987 to 16985, or 26.1% reduction.

The fault coverage for the Boolean circuits only slightly increases with 0.20% from 97.95% to 98.15%, while for the three-state circuits, there is a very significant fault coverage improvement from 94.29% to 96.58%. This means that 40% of the faults that were not covered for the three-state circuits without TPs become covered by ATPG patterns after TPI ($(1 - \frac{(100\%-96.58\%)}{(100\%-94.29\%)}) \cdot 100\% = 40\%$). For all industrial benchmark circuits, the ATPG fault coverage improves from 94.76% without TPs to 96.78% with TPs, or 38.5% of the faults that were not dectected by ATPG patterns without TPs become detected after TPI.

The CPU time spent on compact ATPG is also significantly reduced after TPI. In case of the Boolean industrial circuits, the CPU time is reduced from 6005 seconds to 2731 seconds or 54.5% reduction. For the three-state circuits, the reduction even is 65.1%, from 180180 seconds to 62803 seconds. Especially the reduction for circuit p824184 is remarkable. For this circuit the CPU time spent on ATPG is reduced with 81.2%. Overall industrial benchmark circuits, the CPU time reduction is 64.8%.

5.2 Testability analysis measures for detecting specific ATPG/test set size problems

Besides COP, other TA measures exist that can help to indicate if there are testability problems in the circuit that can cause a large test set. The TA measures *SCOAP* [Gol80] and *test counts (TCs)* [Hay74] can also be used to find testability problems. These TA measures are described in Subsections 5.2.1 and 5.2.2.

5.2.1 SCOAP

Sandia Controllability/Observability Analysis Program (SCOAP) is a TA measure that is used by commercial ATPG tools, i.e., used in AMSAL (Appendix E). Similar to COP, SCOAP divides the measures in controllability and observability measures. The SCOAP controllability of a signal line is an estimate for the number of inputs that have to be assigned in order to get a 0(1) on the line. The SCOAP observability of a line is an estimate for the number of inputs that have to be set to propagate a fault-effect from that line to an output. Besides the number of inputs that have to be set, SCOAP also takes into account how ’deep’(level) the line is in the circuit and the number of fan-out branches that will also be assigned due to the input assignments.

ATPGs try to choose the paths with the smallest SCOAP values to activate faults and to propagate fault effects to POs, because the SCOAP values suggest that these paths require the fewest number of input assignments. When TPI is used to reduce the SCOAP values for lines, the ATPG can generate test patterns with fewer assigned inputs for the faults corresponding to these lines. More test patterns can be compacted into one single test pattern, resulting in a better test set size reduction. Because SCOAP values also take into account the level of signal lines and the number of fan-out branches, they are often far from exact with respect to the necessary input assignments in order to activate faults or propagate fault-effect. However, they give a good indication how difficult it is to control a signal line or to make it observable compared to other signal lines.

5.2.2 Test counts

In [Hay74] and [Kri87], a method is presented to find a lower bound on the test set size for a circuit such that all SAFs can be detected. In order to get this lower bound, the TA measures *test counts (TCs)* are calculated for each line in the circuit. They consist of the following two types of values:

1. The *essential zeros (ones)*, *E0*(*E1*):
   The minimum number of times a line must become 0(1) (during the application of a test set) and be observable on an output, such that all corresponding faults in its fan-in cone can be covered..

2. The *total zeros (ones)*, *T0*(*T1*):
   The minimum total number of times a line must become 0(1) (during the application of a test set) and either be observable or making other lines observable..

Because for each line l, a minimum of $T0_l$+$T1_l$ patterns are necessary ($T0_l$ patterns for the minimum total number of times l must be 0 and $T1_l$ patterns for the minimum total number of times l must be 1), Eq. 5.1 can be used to calculate a lower bound on the minimum test set size, T_{min} necessary for achieving complete fault coverage, in which L represents the number of lines in the circuit.

$$T_{min} \geq max_{l=1}^{L}(T0_l + T1_l) \tag{5.1}$$

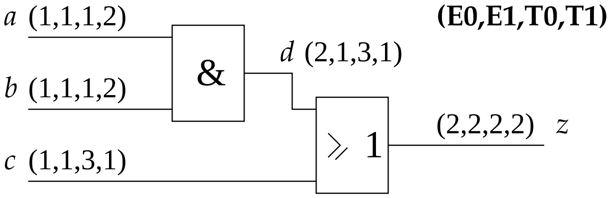


Figure 5.1: Example of test counts (TCs) in a small (fan-out free) circuit

Fig. 5.1 shows a small given circuit. For each line in this circuit, the TCs, i.e., essential zeros, essential ones, total zeros and total ones, are already listed. How these TCs are obtained is explained in the following text.

Input lines do not have a fan-in cone, therefore these lines, i.e., lines *a*, *b*, and *c* only have to be 1 & observable and 0 & observable once to cover for the SA0 and SA1 faults at these lines; $E0_{\{a,b,c\}}$=1 and $E1_{\{a,b,c\}}$=1.

An essential zero at an input of an N-input AND gate is only observable when all other inputs carry the non-dominating value 1. Given the input of the AND gate being 0 & observable, the output of the AND gate will also be 0 & observable. Because all other inputs should carry the non-dominating value 1 in order for an input to be essential zero, the different inputs of the AND gate cannot be essential zero (0 & observable) at the same time. Therefore the essential zero count for the output of an N-input AND gate is the sum of all input essential zeros, see Eq. 5.2,

$$E0_z = \sum_{i=1}^{N}(E0_{x_i}) \qquad \text{AND gate} \tag{5.2}$$

in which $E0_z$ represents the essential zero counts for the output of the N-input AND gate, and $E0_{x_i}$ represents the essential zero count for the ith input of the N-input AND gate. Given the AND gate in Fig. 5.1, the essential zero count of line *d* is 2, i.e., $E0_a$ + $E0_b$ = 1+1 = 2.

An input of an N-input AND gate is only 1 & observable when all other inputs of the AND gate carry the non-dominating value 1. Because the input itself is essential one and all other inputs are also 1, the output of the N-input gate will also be 1 & observable. Because the other inputs should already be 1 in order for an input to be essential one, the N inputs of the AND gate can be essential one at the same time. The essential one count of the output of the N-input AND gate is equal to the largest essential one count found at the inputs of the N-input AND gate, see Eq. 5.3,

$$E1_z = max_{i=1}^{N}(E1_{x_i}) \qquad \text{AND gate} \tag{5.3}$$

in which $E1_z$ represents the essential one counts for the output of the N-input AND gate, and $E1_{x_i}$ represents the essential one count for the ith input of the N-input AND gate.

Given the AND gate in Fig. 5.1, the essential one count of line d is 1, i.e., max(E1_a, E1_b) = max(1,1) = 1.

The the essential zero counts and essential one counts equations for an N-input OR gate are derived in a similar way as for the AND gate and become:

$$E0_z = max_{i=1}^{N}(E0_{x_i}) \qquad \text{OR gate} \tag{5.4}$$

$$E1_z = \sum_{i=1}^{N}(E1_{x_i}) \qquad \text{OR gate} \tag{5.5}$$

Given Eqs. 5.4 and 5.5, the following essential counts for line z are obtained: $E0_z = max(E0_c, E0_d) = max(2,1) = 2$; $E1_z = E1_c + E1_d = 1 + 1 = 2$.

z is a circuit output, for which the total counts are always equal to the essential counts, hence $T0_z$=$E0_z$ and $T1_z$=$E1_z$. In case of an N-input OR gate, the output of the gate can only be 0 when all inputs are 0. Besides that, an input x_i of the OR gate should also carry the non-dominating value 0 when the other inputs should be 1 & observable, i.e., carry essential ones. Given this, the total zero count for an input x_i of an OR gate can be obtained with Eq. 5.6.

$$T0_{x_i} = T0_z + \sum_{j=1, j \neq i}^{N}(E1_{x_j}) \qquad \text{OR gate} \tag{5.6}$$

where $T0_{x_i}$ and $T0_z$ are the total zero counts for OR gate input x_i, respectively OR gate output z, and $E1_{x_j}$ the essential one count for OR gate input x_j. Given Eq. 5.6, line d should not only be 0 when z should be 0, but also when c carries an essential one value; $T0_d = T0_z + E1_c = 2 + 1 = 3$. Same applies to line c, it should not only be 0 when z should be 0, but also when d carries an essential 1 value; $T0_c = T0_z + E1_d = 2 + 1 = 3$.

For an OR gate, it is not necessary for an input to be 1 when the output should be 1; it is allowed to be 0 as long as there is another input that carries a 1. A 1 on an input of the OR gate will also not make other inputs of the OR gate observable, therefore the total one count of an input of an OR gate is just the number of times that the input itself should be 1 & observable. In other words, the total one counts of an input of an OR gate equals the essential one counts of that input, see Eq. 5.7.

$$T1_{x_i} = E1_{x_i} \qquad \text{OR gate} \tag{5.7}$$

Given Eq. 5.7, the total one counts for line c and d become: $T1_c = E1_c = 1$, $T1_d = E1_d = 1$.

The total counts equation for an AND gate are derived in a similar way as for the OR gate and are given in Eqs. 5.8 and 5.9.

$$T0_{x_i} = E0_{x_i} \qquad \text{AND gate} \tag{5.8}$$

$$T1_{x_i} = T1_z + \sum_{j=1, j \neq i}^{N}(E0_{x_j}) \qquad \text{AND gate} \tag{5.9}$$

Given Eqs. 5.8 and 5.9, the total counts for lines a and b are obtained. $T0_a = E0_a = 1$, $T1_a = T1_d + E0_b = 1 + 1 = 2$, $T0_b = E0_b = 1$, $T1_b = T1_d + E0_a = 1 + 1 = 2$.

With Eq. 5.1, the lower bound on the test set size can be obtained. For the example circuit of Fig. 5.1, the found lower bound is 4.

Table 5.3 shows for all input patterns of the circuit given in Fig. 5.1 which faults will be detected. The first three columns, Columns a, b and c represent the input patterns and the remaining ten columns show for all SAFs at lines a, b, c, d and z if the given pattern covers (x) or does not cover (-) the fault. Given the fault coverage matrix of Table 5.3, 3 minimum test sets of 4 patterns covering all faults are found. These three sets are: TS1={`110`, `010`, `100`, `001`}, TS2={`110`, `010`, `100`, `011`}, and TS3={`110`, `010`, `100`, `101`}. In TS1 and TS2, a=0 occurs twice, while in TS3, a=0 only occurs 1. Hence, a should be 0 at least once for complete coverage, which equals the total zero count of a calculated before, $T0_a = 1$. In TS1 and TS2, a=1 occurs twice, while in TS3, a=1 even occurs three times. Hence a should be 1 at least twice for complete coverage, which equals the total one count of a calculated before, $T1_a = 2$. In the same way the correctness of the total counts for lines b and c can be checked. Hence for this example circuit, the TCs are exact and the lower bound of the test set found with Eq. 5.1 equals the actual minimum test set size.

The TC method is only exact for fan-out free circuits (the circuit given in Fig. 5.1 is fan-out free), because it cannot be known in advance how the essential zeros/ones will divide over the different fan-out branches during actual ATPG. Heuristics are necessary to predict how the TCs will divide over these branches. Section 5.3 will describe the heuristic that is used in our proposed TPI algorithm. In [Kri87], propagation of the essential values over branches is ignored: each branch is assigned an *E0*/*E1* value of 1.

Table 5.3: Fault coverage matrix for all fault in Fig. 5.1

Pattern			Stuck-at 0 fault					Stuck-at 1 fault				
a	b	c	a	b	c	d	z	a	b	c	d	z
0	0	0	-	-	-	-	-	-	-	x	x	x
0	0	1	-	-	x	-	x	-	-	-	-	-
0	1	0	-	-	-	-	-	x	-	x	x	x
0	1	1	-	-	x	-	x	-	-	-	-	-
1	0	0	-	-	-	-	-	-	x	x	x	x
1	0	1	-	-	x	-	x	-	-	-	-	-
1	1	0	x	x	-	x	x	-	-	-	-	-
1	1	1	-	-	-	-	x	-	-	-	-	-

When a circuit has lines with very large TC values, it will also have a very large ATPG test set size, because $T0+T1$ gives a lower bound on the necessary test set size in order to cover all SAFs in the circuit. With a CF that takes TCs into account, the TPI algorithm will reduce the TC values. With reduced TC values, also the lower bound on the minimum test set size reduces, which makes smaller test sets possible.

5.3 Test counts in a TPI cost function

For circuits with (reconvergent) fan-out, the TCs are not very accurate, because it is not known how the essential counts are divided over the fan-out branches. Therefore it would not seem to be a good idea to only rely on the individual TCs to find the best TP positions in the circuit in order to get more test set size reduction. Therefore TCs will only be used as a guidance in selecting the right TP positions. In [Geu00], we proposed a CF that takes into account both COP and TC values, such that TPs will be selected on the basis of minimizing a CF, hence both increasing PR detectability and reducing TC values in the circuit. Results in [Geu00] and [Geu01] have shown that by taking into account TCs, better compact ATPG test set size reductions can be achieved, as will be described in this section.

The following text illustrates the basic idea of the CF that takes into account both COP and TCs. Given that the following two lines are part of a circuit: Line *a*, with many hard-to-test RPR faults in its input-cone and through which only one or two tests have to pass, and line *b* with fewer hard-to-test RPR faults in its input-cone but through which many tests have to pass. It is assumed that these two lines are the only two TP candidates.

According to CFs that are only based on COP, e.g., Eqs. 3.11 and 4.30, the TPI algorithm will primarily focus on solving the PR testability problems for line *a* because its faults are the most hard-to-test. But for the ATPG this will mainly result in an easier generation of only a few patterns. These patterns will have fewer assigned inputs, however there are only a few of these patterns such that this will not have too much impact on the total test set size.

The proposed cost method based on TCs and COP however, will insert the TP at line *b*. In spite of the fact that the faults in its input-cone are easier to test pseudo-randomly than those of *a*, the cost contribution of *b* will be much higher because many tests (fault effects) have to pass through *b*. A TP at *b* will probably result in a larger cost reduction than at *a*. For the ATPG this means that many faults will be easier to detect and will have fewer inputs assigned in their tests. When there are many faults that require fewer inputs assigned in order to be detected, the impact on the compaction will be larger.

Subsection 5.3.1 introduces the proposed COP *and* TC based CF and the corresponding cost gradient value equations. Using TC values in the CF has a significant impact on the Hybrid TPI algorithm. The consequences of the TC-based CF on the TPI algorithm are described in Subsection 5.3.2.

5.3.1 TC and COP based cost function

Eq. 5.10 shows the CF used in [Geu00]; it takes into account both COP and TC values and is based on the original HCRF CF for a line *l* [Tsa97], see Eq. 3.14.

$$K_l = \frac{E1_l}{C1_l \cdot W_l} + \frac{T1_l - E1_l}{C1_l} + \frac{E0_l}{C0_l \cdot W_l} + \frac{T0_l - E0_l}{C0_l} \tag{5.10}$$

This CF consists of the following four fractions:

1. The first fraction represents the number of times that line *l* should be essential one, i.e., the minimum number of times that line *l* should be driven to 1 and be observable at the same time, divided by the detection probability of the SA0 fault at *l*, i.e., the probability that line *l* is 1 & observable.
2. The second fraction represents the remaining minimum number of times that line *l* should be driven to 1, i.e., should be 1 but not necessarily be observable (the total one counts minus the essential one counts), divided by the probability that *l* is 1.
3. The third fraction represents the number of times that line *l* should be essential zero, i.e., the minimum number of times that line *l* should be driven to 0 and be observable at the same time, divided by the detection probability of the SA1 fault at *l*, i.e., the probability that line *l* is 0 & observable.
4. The last fraction represents the remaining minimum number of times the line should be driven to 0, i.e., should be 0 but not necessarily be observable (the total zero counts minus the essential zero counts), divided by the probability that *l* is 0.

This CF does not only take into account the probability of lines being 1(0) & observable (i.e., the detection probability), but also takes into account the minimum number of times that they should be 1(0) & observable (i.e., the TCs) to cover all faults in the fan-in cone of the line. In three-state circuits, essential Zs and total Zs can also occur, but there they are very rare, therefore we have not taken them into account.

A new CF means new cost gradient equations that are used by the HCRF TPI algorithm, see Subsections 3.5.3 and 4.2.4. The chain-rule for the cost gradient calculation does not change, but the parts of the equations with respect to the impact of controllability/observability changes on the cost contribution of the line itself do change. Given the CF of Eq. 5.10, Eqs. 5.11-5.12 are the derived gradient equations with respect to observability changes on gate output *z*, and Eqs. 5.14-5.16 are the derived gradient equations with respect to controllability changes on gate input x_j.

$$\frac{dK_z}{dW_z} = -\frac{E0_z}{C0_z \cdot W_z^2} - \frac{E1_z}{C1_z \cdot W_z^2} \tag{5.11}$$

$$\frac{dK_z}{dWZ_z^0} = 0 \tag{5.12}$$

$$\frac{dK_z}{dWZ_z^1} = 0 \tag{5.13}$$

$$\frac{dK_{x_j}}{dC0_{x_j}} = -\frac{E0_{x_j}}{C0_{x_j}^2 \cdot W_{x_j}} + \frac{E0_{x_j} - T0_{x_j}}{C0_{x_j}^2} \tag{5.14}$$

$$\frac{dK_{x_j}}{dC1_{x_j}} = -\frac{E1_{x_j}}{C1_{x_j}^2 \cdot W_{x_j}} + \frac{E1_{x_j} - T1_{x_j}}{C1_{x_j}^2} \tag{5.15}$$

$$\frac{dK_{x_j}}{dCZ_{x_j}} = 0 \tag{5.16}$$

$\mathrm{d}K/\mathrm{d}CZ$, $\mathrm{d}K/\mathrm{d}WZ^0$ and $\mathrm{d}K/\mathrm{d}WZ^1$ are always 0, because this CF does not take into account Z controllabilities, Z↔0 observabilities and Z↔1 observabilities. However, in three-state circuits, there will be non-zero Z controllabilities, Z↔0 or Z↔1 observabilities. Therefore the COP calculations and the cost gradient chain-rule will still have to take into account these Z probabilities.

Because the CF does take into account TC measures and so will change when the TC measures change, there also exist cost gradients with respect to essential counts and total counts. The total counts cost gradients are derived in a similar way as the COP observability cost gradients, see Subsection 3.5.3; they consist of a part with respect to the total counts changes on a gate output z itself and a chain-rule for the total counts changes on the gate inputs caused by the changes at z. Eqs. 5.17 and 5.18 show the total counts cost gradients,

$$\frac{dK}{dT0_z} = \frac{dK_z}{dT0_z} + \sum_{i=1}^{X} \left(\frac{dK}{dT0_{x_i}} \frac{dT0_{x_i}}{dT0_z} + \frac{dK}{dT1_{x_i}} \frac{dT1_{x_i}}{dT0_z} \right) \tag{5.17}$$

$$\frac{dK}{dT1_z} = \frac{dK_z}{dT1_z} + \sum_{i=1}^{X} \left(\frac{dK}{dT0_{x_i}} \frac{dT0_{x_i}}{dT1_z} + \frac{dK}{dT1_{x_i}} \frac{dT1_{x_i}}{dT1_z} \right) \tag{5.18}$$

Given CF 5.10, the first part of the total counts cost gradients become:

$$\frac{dK_z}{dT0_z} = \frac{1}{C0_z} \tag{5.19}$$

$$\frac{dK_z}{dT1_z} = \frac{1}{C1_z} \tag{5.20}$$

and the total counts cost gradients become:

$$\frac{dK}{dT0_z} = \frac{1}{C0_z} + \sum_{i=1}^{X} \left(\frac{dK}{dT0_{x_i}} \frac{dT0_{x_i}}{dT0_z} + \frac{dK}{dT1_{x_i}} \frac{dT1_{x_i}}{dT0_z} \right) \tag{5.21}$$

$$\frac{dK}{dT1_z} = \frac{1}{C1_z} + \sum_{i=1}^{X} \left(\frac{dK}{dT0_{x_i}} \frac{dT0_{x_i}}{dT1_z} + \frac{dK}{dT1_{x_i}} \frac{dT1_{x_i}}{dT1_z} \right) \tag{5.22}$$

The essential counts cost gradients are derived in a similar way as the COP controllability cost gradients, see Subsection 3.5.3; they consist of a part with respect to the essential counts changes on the gate input x_j itself, a chain-rule for the essential counts changes on the Z gate output(s) caused by the changes at x_j, and a chain rule for the total counts changes on the other $X-1$ inputs of the gate caused by the essential counts changes on x_j. Eqs. 5.23 and 5.24 show the essential counts cost gradients.

$$\begin{aligned}\frac{dK}{dE0_{x_j}} &= \frac{dK_{x_j}}{dE0_{x_j}} + \sum_{i=1,i\neq j}^{X}\left(\frac{dK}{dT0_{x_i}}\frac{dT0_{x_i}}{dE0_{x_j}} + \frac{dK}{dT1_{x_i}}\frac{dT1_{x_i}}{dE0_{x_j}}\right) \\ &\quad + \sum_{k=1}^{Z}\left(\frac{dK}{dE0_{z_k}}\frac{dE0_{z_k}}{dE0_{x_j}} + \frac{dK}{dE1_{z_k}}\frac{dE1_{z_k}}{dE0_{x_j}}\right)\end{aligned} \tag{5.23}$$

$$\begin{aligned}\frac{dK}{dE1_{x_j}} &= \frac{dK_{x_j}}{dE1_{x_j}} + \sum_{i=1,i\neq j}^{X}\left(\frac{dK}{dT0_{x_i}}\frac{dT0_{x_i}}{dE1_{x_j}} + \frac{dK}{dT1_{x_i}}\frac{dT1_{x_i}}{dE1_{x_j}}\right) \\ &\quad + \sum_{k=1}^{Z}\left(\frac{dK}{dE0_{z_k}}\frac{dE0_{z_k}}{dE1_{x_j}} + \frac{dK}{dE1_{z_k}}\frac{dE1_{z_k}}{dE1_{x_j}}\right)\end{aligned} \tag{5.24}$$

Given CF 5.10, the first part of the essential counts cost gradients become:

$$\frac{dK_{x_j}}{dE0_{x_j}} = \frac{(1-W_{x_j})}{C0_{x_j}\cdot W_{x_j}} \tag{5.25}$$

$$\frac{dK_{x_j}}{dE1_{x_j}} = \frac{(1-W_{x_j})}{C1_{x_j}\cdot W_{x_j}} \tag{5.26}$$

and the essential counts cost gradients become:

$$\begin{aligned}\frac{dK}{dE0_{x_j}} &= \frac{(1-W_{x_j})}{C0_{x_j}\cdot W_{x_j}} + \sum_{i=1,i\neq j}^{X}\left(\frac{dK}{dT0_{x_i}}\frac{dT0_{x_i}}{dE0_{x_j}} + \frac{dK}{dT1_{x_i}}\frac{dT1_{x_i}}{dE0_{x_j}}\right) \\ &\quad + \sum_{k=1}^{Z}\left(\frac{dK}{dE0_{z_k}}\frac{dE0_{z_k}}{dE0_{x_j}} + \frac{dK}{dE1_{z_k}}\frac{dE1_{z_k}}{dE0_{x_j}}\right)\end{aligned} \tag{5.27}$$

$$\begin{aligned}\frac{dK}{dE1_{x_j}} &= \frac{(1-W_{x_j})}{C1_{x_j}\cdot W_{x_j}} + \sum_{i=1,i\neq j}^{X}\left(\frac{dK}{dT0_{x_i}}\frac{dT0_{x_i}}{dE1_{x_j}} + \frac{dK}{dT1_{x_i}}\frac{dT1_{x_i}}{dE1_{x_j}}\right) \\ &\quad + \sum_{k=1}^{Z}\left(\frac{dK}{dE0_{z_k}}\frac{dE0_{z_k}}{dE1_{x_j}} + \frac{dK}{dE1_{z_k}}\frac{dE1_{z_k}}{dE1_{x_j}}\right)\end{aligned} \tag{5.28}$$

5.3.2 Implications of a TC based CF on the TPI algorithm

As mentioned in Subsection 5.2.2, TCs are only exact in fan-out free circuits, because it is not known how TC values will propagate over fan-out branches. In general all circuits have fan-out branches, therefore before TCs can be used in the CF of the Hybrid TPI algorithm, a heuristic must be established to divide the TC values over the different fan-out branches. As mentioned in Subsection 5.2.1, ATPGs use SCOAP values to choose which paths to take to activate or propagate fault effects. The lower the SCOAP value of a line, the higher the probability that the ATPG will use that line to activate faults or propagate fault effects. Therefore in [Geu00] we proposed to use a SCOAP based heuristic to divide the essential counts over fan-out branches. The higher the SCOAP observability of a fan-out branch, the less likely that essential counts will propagate over that fan-out branch. The branch with the lowest SCOAP observability is assumed to propagate most of the essential counts. Given a fan-out stem x with L fan-out branches z_l, with each fan-out branch having a SCOAP observability SW_{z_l}, we use the following heuristics for propagating the essential counts over the fan-out branches:

$$E0_{z_l} = \frac{\frac{1}{SW_{z_l}}}{\sum_{i=1}^{L}\frac{1}{SW_i}} \cdot E0_x \tag{5.29}$$

$$E1_{z_l} = \frac{\frac{1}{SW_{z_l}}}{\sum_{i=1}^{L}\frac{1}{SW_i}} \cdot E1_x \tag{5.30}$$

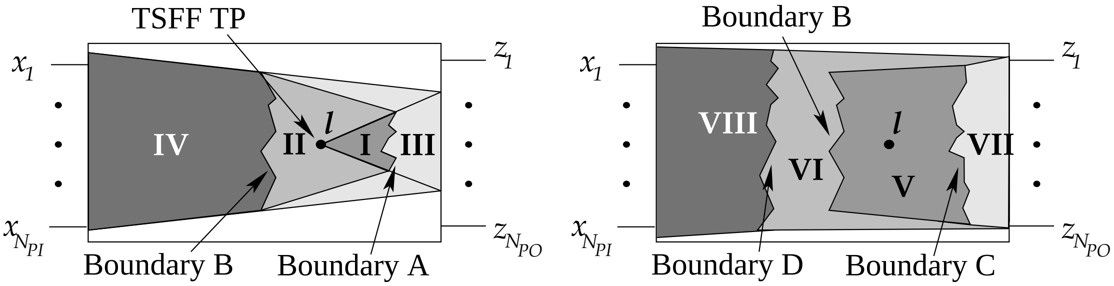
Because its more likely that essential values propagate over lines with smaller SCOAP observability, we use $\frac{1}{SW}$, such that $\frac{1}{SW_{z_l}}$ is larger for branches z_l with smaller SCOAP values. The normalization with $\sum_{i=1}^{L}\frac{1}{SW_i}$ is used such that the sum of the essential accounts over all branches does not exceed the essential count of the fan-out stem.

The event-driven algorithm to calculate the HCRF values for each TP candidate becomes more complicated due to the TC based CF. Because the essential counts in the circuit depend on the SCOAP observabilities, i.e., the propagation of the essential counts over fan-out branches depends on the SCOAP observability, an extra forward and backward propagation of cost changes is necessary in the event-driven HCRF algorithm, as is explained in the following text.

The HCRF calculation given a TC based CF, like Eq. 5.10, can be divided into two steps. The first step consists of the HCRF calculation for a non TC based CF, e.g., Eqs. 3.11 and 4.30, as described in Subsection 4.3.1. In addition to the HCRF calculation for a non TC based CF, an extra step is necessary to take into account essential counts changes in the circuit due to SCOAP observability changes. Descriptions of these two steps are given in the following text:

Step 1 of the HCRF calculation given a TC based cost function

This first part of the calculation is depicted in Fig. 5.2(a). This figure is the same as Fig. 4.3, which describes the HCRF calculation given a non TC based CF. The impact of the TP on the TA values, and hence on the cost reduction, differs for the four differently



(a) Propagation of TA measures changes

(b) Propagation of TC changes due to SCOAP observability changes

Figure 5.2: Computing HCRF given a TC based cost function

shaded regions and is zero for the remaining (white) part of the circuit. These shaded regions and their different impacts are:

Region I: The part of the circuit in the fan-out cone of line *l* up to Boundary A in which the TSFF TP will cause significant changes in controllabilities and/or essential counts.

Region II: The part of the circuit between Boundary A and Boundary B in which the TSFF TP will cause significant observabilities and/or total counts changes. Region II includes Region I.

Region III: The part of the circuit between Boundary A and the POs in which the TSFF TP will only cause small changes in controllabilities, observabilities, essential counts and total counts.

Region IV: The part of the circuit between the PIs and Boundary B in which the TSFF TP will only cause small changes in observabilities and total counts.

Related to Regions I-IV, the first step of the HCRF for a TSFF TP at line *l* comprises the following four parts:

1. $\sum_f \left(K_f^{(Org)} - K_f^{(m)}\right)$ *for every fault f inside Regions I and II.*
   The TA measures (COP, SCOAP and TC) of the lines corresponding to the faults in Regions I and II and therefore the cost contributions of the faults are computed explicitly.

2. $\sum_{l_{bA} \in Boundary\ A} \left(\frac{dK}{dC0_{l_{bA}}} \cdot \Delta C0_{l_{bA}} + \frac{dK}{dC1_{l_{bA}}} \cdot \Delta C1_{l_{bA}} \right) +$
   $\sum_{l_{bA} \in Boundary\ A} \left(\frac{dK}{dE0_{l_{bA}}} \cdot \Delta E0_{l_{bA}} + \frac{dK}{dE1_{l_{bA}}} \cdot \Delta E1_{l_{bA}} \right)$ *for every fault in Region III.*

The COP controllability and essential counts changes of the lines l_{bA} on Boundary A ($\Delta C0_{l_{bA}}$, $\Delta C1_{l_{bA}}$, $\Delta E0_{l_{bA}}$ and $\Delta E1_{l_{bA}}$) and the COP controllability and essential counts gradients of these lines are used to estimate the impact of the TP on the cost contribution of the faults in Region III, i.e., $\sum_{f \in Region\ III} \left(K_f^{(Org)} - K_f^{(m)} \right)$.

3. $\sum_{l_{bB} \in Boundary\ B} \left(\frac{dK}{dW_{l_{bB}}} \cdot \Delta W_{l_{bB}} + \frac{dK}{dT0_{l_{bB}}} \cdot \Delta T0_{l_{bB}} + \frac{dK}{dT1_{l_{bB}}} \cdot \Delta T1_{l_{bB}} \right)$ *for every fault in Region IV.*
   The COP observability and total counts changes of the lines l_{bB} on Boundary B ($\Delta W_{l_{bB}}$, $\Delta T0_{l_{bB}}$ and $\Delta T1_{l_{bB}}$) and the COP observability and total counts gradients of these lines are used to estimate the impact of the TP on the cost contribution of the faults in Region IV, i.e., $\sum_{f \in Region\ IV} \left(K_f^{(Org)} - K_f^{(m)} \right)$.

4. *The cost contribution of the faults at the inserted TSFF output.*
   The contribution of the new faults introduced by the inserted TSFF TP are added as negative contribution to the cost reduction: $\frac{1}{Pd_{l'/SA0}}$ and $\frac{1}{Pd_{l'/SA1}}$. These are the SA0/SA1 faults at the TSFF output, see Fig. 3.4(d).

Step 2 of the HCRF calculation given a TC based cost function

The SCOAP observability changes in Regions II (including I) and IV have as a result that the essential counts will be distributed differently over the fan-out branches, see Eqs. 5.29 and 5.30. Therefore, for each fan-out branch in Region II where the SCOAP observability has changed, the impact of the altered essential count distribution on the cost has to be recalculated. This recalculation is also performed using the hybrid event driven algorithm and is visualized in Fig. 5.2(b). The impact of the SCOAP observability changes, caused by the inserted TSFF TP, on the essential and total counts is different for each shaded region in Fig. 5.2(b), and is zero for the remaining (white) part of the circuit. These shaded regions are:

Region V: The part of the circuit in the fan-out cone of Boundary B up to Boundary C in which the SCOAP observability changes will result in significant changes in essential counts.

Region VI: The part of the circuit between Boundary C and Boundary D in which the SCOAP observability changes will cause significant changes in total counts. Region VI includes Region V.

Region VII: The part of the circuit between Boundary C and the POs, in which the SCOAP observability changes will only cause small changes in essential and total counts.

Region VIII: The part of the circuit between the PIs and Boundary D, in which the TSFF TP will only cause small changes in total counts.

Related to Regions V-VIII, the second step of the HCRF for a TSFF TP at line l comprises the following three parts:

5. $\sum_f \left(K_f^{(m)} - K_f^{(m')}\right)$ *for every fault f inside Regions V and VI.*
   The TCs of the lines corresponding to the faults in Regions V and VI and therefore the cost contributions of these faults are computed explicitly. $K_f^{(m)}$ is the cost contribution of fault f in the circuit with the TP candidate when the TC values have *not yet* been updated for to SCOAP observability changes and $K_f^{(m')}$ is the cost contribution of fault f in the circuit with the TP candidate in case the TC values have been updated for the SCOAP observability changes.

6. $\sum_{l_{bC} \in Boundary\ C} \left(\frac{dK}{dE0_{l_{bC}}} \cdot \Delta E0_{l_{bC}} + \frac{dK}{dE1_{l_{bC}}} \cdot \Delta E1_{l_{bC}}\right)$ *for every fault in Region VII.*
   The essential counts changes of the lines l_{bC} on Boundary C and the essential counts gradients of these lines are used to estimate the impact of the SCOAP observability changes on the cost contribution of the faults in Region VII, i.e., $\sum_{f \in Region\ VII} \left(K_f^{(m)} - K_f^{(m')}\right)$.

7. $\sum_{l_{bD} \in Boundary\ D} \left(\frac{dK}{dT0_{l_{bD}}} \cdot \Delta T0_{l_{bD}} + \frac{dK}{dT1_{l_{bD}}} \cdot \Delta T1_{l_{bD}}\right)$ *for every fault in Region VIII.*
   The total counts changes of the lines l_{bD} on Boundary D and the total counts gradients of these lines are used to estimate the impact of SCOAP observability changes on the cost contribution of the faults in Region VIII, i.e., $\sum_{f \in Region\ VIII} \left(K_f^{(m)} - K_f^{(m')}\right)$.

Combining the 7 parts of Step 1 and 2 in one equation, the HCRF estimate for a TSFF TP at line l given a TC based CF becomes:

$$\begin{aligned}
HCRF_l = & \sum_{f \in Reg.\ I\&II} \left(K^{(Org)} - K^{(m)}\right) \\
& - \sum_{l_{bA} \in Bound.\ A} \left(\frac{dK}{dC0_{l_{bA}}} \cdot \Delta C0_{l_{bA}} + \frac{dK}{dC1_{l_{bA}}} \cdot \Delta C1_{l_{bA}} + \frac{dK}{dE0_{l_{bA}}} \cdot \Delta E0_{l_{bA}} + \frac{dK}{dE1_{l_{bA}}} \cdot \Delta E1_{l_{bA}}\right) \\
& - \sum_{l_{bB} \in Bound.\ B} \left(\frac{dK}{dW_{l_{bB}}} \cdot \Delta W_{l_{bB}} + \frac{dK}{dT0_{l_{bB}}} \cdot \Delta T0_{l_{bB}} + \frac{dK}{dT1_{l_{bB}}} \cdot \Delta T1_{l_{bB}}\right) \\
& - \left(K_{l'/SA0} + K_{l'/SA1}\right) \\
& + \sum_{f \in Reg.\ V\&VI} \left(K^{(m)} - K^{(m')}\right) \\
& - \sum_{l_{bC} \in Bound.\ C} \left(\frac{dK}{dE0_{l_{bC}}} \cdot \Delta E0_{l_{bC}} + \frac{dK}{dE1_{l_{bC}}} \cdot \Delta E1_{l_{bC}}\right) \\
& - \sum_{l_{bD} \in Bound.\ D} \left(\frac{dK}{dT0_{l_{bD}}} \cdot \Delta T0_{l_{bD}} + \frac{dK}{dT1_{l_{bD}}} \cdot \Delta T1_{l_{bD}}\right)
\end{aligned} \tag{5.31}$$

Table 5.4: Compact ATPG results of ISCAS circuits after TC&COP TPI

Circuit	*NPAT* TPI			TPI		TC&COP TPI		
	T	FC(%)	CPU	TSFF	CPU	T	FC(%)	CPU
c1908	36	99.92	0.46 s	25	1.75 s	42	99.92	0.46 s
c3540	75	97.80	1.74 s	25	4.39 s	73	97.67	1.69 s
c7552	60	99.47	3.16 s	40	8.29 s	74	99.40	3.40 s
Subtotal	171	99.08	5.36 s	90	14.4 s	189	99.01	5.55 s
s1196	45	100.00	0.30 s	20	0.86 s	52	100.00	0.30 s
s1488	90	100.00	0.48 s	20	1.19 s	88	100.00	0.44 s
s1494	89	99.87	0.49 s	20	1.22 s	87	99.87	0.46 s
s5378	57	98.93	1.56 s	30	2.86 s	60	98.91	1.53 s
s9234.1	53	94.85	4.46 s	50	10.8 s	51	94.69	4.24 s
s13207.1	223	99.22	11.9 s	50	16.4 s	202	99.16	11.0 s
s15850.1	98	98.30	10.1 s	50	15.9 s	88	98.38	9.72 s
s38584.1	122	97.33	39.7 s	50	33.3 s	110	96.53	36.4 s
Subtotal	777	97.76	68.9 s	290	82.6 s	738	97.39	64.0 s
s499a	31	100.00	0.19 s	8	0.21 s	28	100.00	0.18 s
s938a	40	100.00	0.25 s	12	0.54 s	40	100.00	0.25 s
s3330a	61	100.00	0.99 s	32	2.20 s	67	100.00	1.03 s
Subtotal	132	100.00	1.42 s	52	2.95 s	135	100.00	1.46 s
Total	1080	98.03	75.7 s	432	100 s	1062	97.71	71.1 s

5.3.3 Results of HCRF TPI with TC based cost function

Tables 5.4 and 5.5 give a comparison between *NPAT* TPI and the proposed TC and COP based CF, i.e., TC&COP TPI, of the compact ATPG test set sizes after TPI on the ISCAS benchmark circuits, respectively the industrial benchmark circuits [Geu00].

Column *NPAT TPI* shows the ATPG results for the circuits listed in Column *Circuit* after *NPAT* TPI; they consist of the number of ATPG test patterns (*T*), the ATPG fault coverage (*FC(%)*) and the CPU time spent on ATPG (*CPU*). Column *TPI* shows the number of inserted TSFF TPs (*TSFF*) and the CPU time spent on TC&COP TPI (*CPU*). Column *TC&COP TPI* shows the ATPG results for the circuit after TPI with the TC&COP based CF; again the number of ATPG test patterns (*T*), the ATPG fault coverage (*FC(%)*) and the CPU time spent on ATPG (*CPU*) are listed.

The results in Table 5.4 show for the ISCAS'85 circuits that the TC& COP based CF does result in less ATPG test set size reduction than the *NPAT* reduction. Especially for circuit c7552, 14 more ATPG patterns (74 versus 60) are generated after TC&COP TPI compared to *NPAT* TPI. These circuits do not suffer from high TCs, therefore using a CF that takes into account TCs does not seem to be a very good idea.

On the other hand, the compact ATPG test set size reduction for the ISCAS'89 circuits

Table 5.5: Compact ATPG results of industrial circuits after TC&COP TPI

	NPAT TPI			TPI		TC&COP TPI		
Circuit	T	FC(%)	CPU	TSFF	CPU	T	FC(%)	CPU
p7653	153	99.63	8.25 s	15	5.06 s	178	99.62	8.74 s
p14148	218	100.00	17.4 s	15	6.43 s	260	99.99	19.9 s
p27811	168	95.29	32.7 s	38	18.9 s	97	95.22	25.2 s
p31025	449	98.67	129 s	25	45.1 s	463	98.87	129 s
p34592	162	99.98	56.1 s	34	21.3 s	198	99.98	75.4 s
p36503	110	97.22	37.1 s	30	21.4 s	159	97.25	48.1 s
p43282	207	97.70	123 s	49	59.2 s	197	97.73	127 s
p43663	98	99.25	36.0 s	50	28.7 s	132	99.25	40.1 s
p72767	311	96.65	409 s	50	159 s	302	97.25	404 s
p73133	301	96.73	405 s	50	148 s	285	97.39	404 s
p73257	2132	98.06	1030 s	73	141 s	1308	98.09	561 s
p75344	196	99.25	114 s	55	72.1 s	174	99.31	102 s
p162057	556	98.78	330 s	162	432 s	447	98.69	348 s
Subtotal	5061	98.15	2731 s	646	1159 s	4200	98.28	2296 s
p32118	263	93.53	39.8 s	32	22.3 s	147	93.56	30.2 s
p37021	136	77.28	21.3 s	37	27.5 s	176	77.28	23.2 s
p66171	1792	96.79	580 s	66	465 s	1177	96.79	354 s
p71553	76	96.61	395 s	71	204 s	96	96.86	290 s
p93140	367	94.25	616 s	93	228 s	325	94.03	587 s
p104649	785	99.37	394 s	104	203 s	476	99.39	367 s
p114605	513	98.64	520 s	80	220 s	366	99.01	500 s
p137498	276	98.82	199 s	137	315 s	274	98.86	177 s
p481470	1671	99.25	12400 s	185	1686 s	1435	99.28	8390 s
p596922	2100	96.89	9475 s	317	2980 s	521	96.97	2595 s
p598004	1942	98.77	3006 s	100	926 s	2069	98.74	4313 s
p705050	530	91.47	4942 s	250	2630 s	286	76.86	6817 s
p824184	1114	96.73	24468 s	300	4536 s	759	96.73	18655 s
p854266	359	97.62	5741 s	300	4273 s	411	97.65	5442 s
Subtotal	11924	96.58	62803 s	2072	18720 s	8518	94.42	48546 s
Total	16985	96.78	65535 s	2718	19879 s	12718	94.92	50842 s

becomes better with the TC&COP CF. Especially the larger three circuits, i.e., circuits s13207.1, s15850.1 and s38584.1, seem to benefit from the new TC&COP CF. TPI with this CF results in 39 fewer test patterns (738 versus 777) than *NPAT* TPI. The difference in ATPG test set size reduction for the ISCAS'89 addendum circuits is limited. ATPG after TC&COP TPI results in 3 more patterns (135 versus 132) than after *NPAT* TPI.

In total over all ISCAS circuits, the ATPG test set size reduction after TC&COP TPI results in 18 (1080-1062) fewer patterns than *NPAT* TPI. Hence, the TC&COP based CF only results in a small test set size reduction; 18 patterns reduction of 1080 pattern in total is only 1.7%. Most of the ISCAS circuits do not suffer from large TC values, therefore the TC values in the TC&COP based CF do not have a large impact on the TP selection.

Comparing the fault coverages of *NPAT* TPI and TC&COP TPI shows that the ATPG fault coverage after TPI is less for TC&COP TPI (97.71%) compared to *NPAT* TPI (98.03%). For the ISCAS circuits, the *NPAT* TPI results in more faults becoming detectable than TC&COP TPI. On the other hand, the CPU time spent on compact ATPG after TC&COP TPI is less than the CPU time spent on ATPG after *NPAT* TPI. Overall, both CFs are well matched and it is hard to say that one CF performs better on the ISCAS circuits than the other one as this depends on the importance of the ATPG CPU time, the ATPG fault coverage and the number of ATPG patterns.

Contrary to the results of the ISCAS comparison, significant differences are found for the industrial benchmark comparison shown in Table 5.5. For the Boolean industrial benchmark circuits, compact ATPG after TC&COP TPI results in 4200 patterns, while compact ATPG after *NPAT* TPI results in 5061 patterns. In other words, TC&COP results in $\frac{5061-4200}{5061} \cdot 100\% = 17\%$ better test set size reduction than *NPAT* TPI. Especially the ATPG test set size reduction for circuit p73257 is remarkably better with TC&COP TPI (1308 patterns) than with *NPAT* TPI (2132 patterns). This circuit contains many large FFRs which do result in large TCs, see Section 5.4. Using a TC based CF, i.e., TC&COP TPI, seems to be very useful in order to reduce the ATPG test set by reducing the TC values.

The ATPG fault coverage with TC&COP TP is also somewhat better than with *NPAT* TPI (98.28% for TC&COP versus 98.15% for *NPAT* TPI). And finally, for the Boolean industrial circuits, also the CPU time spent on ATPG is significantly less with TC&COP TPI (2296 seconds) than with *NPAT* TPI (2731 seconds); TC&COP TPI is $\frac{2731-2296}{2731} \cdot 100\% = 16\%$ faster than *NPAT* TPI. For the Boolean industrial circuits, the TC&COP TPI outperforms *NPAT* TPI in ATPG test set size reduction, fault coverage and ATPG CPU time reduction.

Also for the three-state industrial circuits, TC&COP TPI results in far better test set size reduction than *NPAT* TPI; TC&COP TPI results in $\frac{11924-8518}{11924} \cdot 100\% = 28.5\%$ better test set size reduction than *NPAT* TPI. Especially the ATPG test set size for circuit p596922 is much smaller after TC&COP TPI than after *NPAT* TPI, 521 versus 2100 patterns. This circuit does not suffer from high TC values. The COP detectabilities in this circuit are very low and the *NPAT* algorithm is not able to handle these poor COP detectabilities very well, as will be explained in Subsection 5.5.1. The test set size reduction for circuit p598004 is still limited after TC&COP TPI, it is even worse compared to *NPAT* TPI. The test set size reduction for this circuit is less than 3%, from 2129 without TPs (see Table 5.2) down to 2069 patterns after TPI. This circuit has one very large FFR that causes the large number of test patterns, as will be explained in Section 5.4.

The difference in ATPG fault coverage after TPI for the three-state industrial circuits, is very large. *NPAT* TPI results in 96.78% ATPG fault coverage, while TC&COP TPI

Table 5.6: TC&COP cost values for faults f_1 and f_2

	f_1	f_2
C0	10^{-3}	10^{-5}
W	0.1	10^{-5}
E0	756	1
T0	512	1
K	$5.4\ 10^{6}$	10^{10}

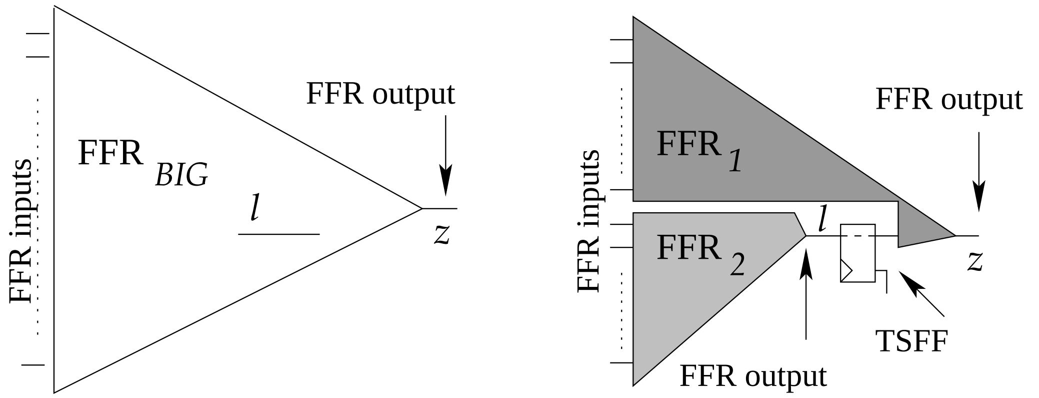
results in only 94.42% reduction. When analysing the individual fault coverages of the three-state industrial circuits, we found that this difference is totally caused by circuit p705050. After *NPAT* TPI 530 ATPG patterns are generated resulting in a fault coverage of 91.47%. TC&COP TPI results in a much smaller ATPG test set, i.e., 286 patterns, but the ATPG fault coverage is only 76.86%. But after an analysis of the fault efficiency for this circuit (not shown in the table due to space limitation), we found that both TC&COP TPI and *NPAT* TPI resulted in 99.9% fault efficiency. In other words, *NPAT* TPI results in many redundant/untestable faults becoming testable after TPI, while TC&COP TPI hardly results in redundant/untestable faults becoming testable after TPI (the fault coverage and efficiency for circuit p705050 without TPs is 76.76%, respectively 99.90%, see Table 5.2). Not taking into account circuit p705050, for eight of the thirteen remaining three-state industrial circuits, the fault coverage after TC&COP TPI is higher than after *NPAT* TPI, while only for two circuits, i.e., circuits p93140 and p598004, *NPAT* TPI results in higher fault coverage than TC&COP TPI.

For the three-state industrial circuits, the CPU time spent on ATPG after TC&COP TPI is significantly less than after *NPAT* TPI, i.e., 48546 seconds versus 62803 seconds. In other words, TC&COP TPI results in $\frac{62803-48546}{62803} \cdot 100\% = 22.7\%$ CPU time reduction compared to *NPAT* TPI.

In total over all industrial benchmark circuits, TC&COP TPI results in significantly better compact ATPG test set size reduction than *NPAT* TPI, i.e., $\frac{16985-12718}{16985} \cdot 100\% = 25.1\%$ better test set size reduction, and also in significantly better compact ATPG CPU time reduction, i.e., $\frac{65535-50842}{65535} \cdot 100\% = 22.4\%$. The fault coverage after *NPAT* TPI is better than after TC&COP TPI, i.e., 96.78% versus 94.42%. However, this is almost totally caused by circuit p705050. Not taking into account this circuit, the difference between the fault coverages after TC&COP TPI and *NPAT* TPI will be small and probably in the advantage of TC&COP TPI.

5.4 TPI and circuits with large Fan-out Free Regions

The experimental results given in Tables 5.1-5.5 have shown that for several circuits (e.g., for circuit p598004) TPI did not result in a significant reduction of the compact



(a) FFR_{BIG} with a very large number of inputs

(b) Two FFRs, FFR_1 and FFR_2 with fewer inputs

Figure 5.3: Splitting of an FFR

ATPG test set sizes, even with the TC&COP based CF. These problem circuits often contain very large *Fan-out Free Regions (FFRs)*. Analysis of the inserted TPs by the TPI algorithm has shown that the TPI algorithm did not insert TPs within these large FFRs. A circuit with very large FFRs will very likely suffer from high TCs and hence large test sets. All faults in that FFR have to pass through the single output of the FFR. Many test patterns are required to make sure that all faults in the FFR are covered by a test set, which can have a significant impact on the total size of the ATPG generated test set. The TC&COP CF does take into account TC measures, however the COP measures dominate the CF. This is explained in the following text. Given a circuit with two SA1 faults f_1 and f_2, for which the TA measures and cost contributions (using the CF of Eq. 5.10) are given in Table 5.6. Although fault f_1 has very high TC values, the cost contribution of fault f_2 is four magnitudes in size larger due to the much smaller COP detectability of fault f_2, such that the TPI algorithm will mainly focus on improving the TA measures for fault f_2 instead of for fault f_1.

The high TCs can be reduced by TPI, which splits these large FFRs into two FFRs with fewer inputs. This is achieved by inserting a TSFF within the FFR and is illustrated in Fig. 5.3. The FFR FFR_{BIG} of Fig. 5.3(a) is split into two smaller FFRs in Fig. 5.3(b), FFR_1 and FFR_2, by the TSFF at line *l*. In this example, the TSFF almost reduces the FFR sizes into half the size of the original FFR_{BIG}. In [Geu02b], we proposed four techniques that can be used to insert TPs in large FFRs. They are described in Subsection 5.4.1. Experimental results of the four techniques to reduce large FFRs are given in Subsection 5.4.2.

5.4.1 Four TPI techniques for reducing large FFRs

The TPI for ATPG algorithm is extended with a TPI pre-process for reducing very large FFRs in the circuit. This pre-process should be run before normal TPI using the Hybrid TPI algorithm (with the TC&COP CF). While the Hybrid CRF TPI algorithm is a global optimization TPI method, the TPI pre-process for reducing large FFRs will be a local optimization method. Only the large FFR testability problem is targeted. The TPI pre-process searches for very large FFRs in the circuit and insert TPs in the large FFRs to reduce them. In the following text four techniques are described that can be used to find and split the FFRs. The size of an FFR corresponds to the number of inputs of the FFR.

The first question which has to be answered is the definition of a ’large FFR’. We use two definitions for a large FFR:

A: An FFR is large when it has more than `LARGE_FFR_THRESHOLD` inputs. `LARGE_FFR_THRESHOLD` is a user-defined threshold.

B: An FFR is large when it has more than $\overline{FFR}$ + 3*σ^{FFR} inputs. $\overline{FFR}$ is the average size of an FFR in the circuit and σ^{FFR} is the standard deviation of the FFR size distribution. Because most of the FFRs only have a small number of inputs, $\overline{FFR}$ and σ^{FFR} are only determined for the larger FFRs ($\geq$ 10 inputs).

Definition **A** uses a fixed, user-defined threshold to determine whether an FFR is very large or not, while definition **B** uses statistics to determine if an FFR is too large.

As already described, large FFRs usually have high TCs. This has led to two strategies to reduce the FFR sizes:

1: The TP candidate that splits the FFR into two smaller FFRs of which the number of inputs are closest to each other (i.e., two approximately equal sized FFRs), is the TP that will be inserted in the FFR.

2: The TP candidate that splits the FFR into two smaller FFRs with both the lowest TC values, is the TP that will be inserted.

The two definitions of large FFRs and the two strategies are combined into four techniques to reduce large FFRs, i.e., methods

A1: A TP will be inserted in the largest FFR, with at least `LARGE_FFR_THRESHOLD` inputs, that results in two smaller (approximately) equal sized FFRs.

A2: A TP will be inserted in the largest FFR, with at least `LARGE_FFR_THRESHOLD` inputs, that results in two smaller FFRs with both the lowest TC values.

B1: A TP will be inserted in the largest, with at least $\overline{FFR}$ + 3*σ^{FFR} inputs, that results in two smaller (approximately) equal sized FFRs.

B2: A TP will be inserted in the largest, with at least $\overline{FFR}$ + 3*σ^{FFR} inputs, that results in two smaller FFRs with both the lowest TC values.

Table 5.7: ATPG results after TPI for reducing large FFRs

			Method A1		Method A2		Method B1		Method B2	
Circuit	TSFF_{TOT}	$\mathrm{T}_{No\ FFR}$	TSFF	T	TSFF	T	TSFF	T	TSFF	T
p73257	73	1308	43	979	43	965	0	1308	0	1308
p162057	162	447	21	204	21	204	2	361	2	361
p32118	32	147	1	133	1	133	1	133	1	133
p93140	93	325	3	334	3	334	3	334	3	334
p104649	104	476	3	510	3	529	4	501	4	518
p137498	137	274	3	276	3	276	3	276	3	276
p481470	185	1435	111	1656	111	1705	38	1428	38	1468
p596922	317	521	20	521	20	517	23	553	23	560
p598004	100	2069	23	1320	23	1319	30	1347	30	1345
p705050	250	286	1	281	1	288	10	284	2	270
p824184	300	759	29	742	0	743	0	895	0	865
p854266	300	411	3	378	3	384	20	388	21	387
Total	2053	8458	261	7334	232	7397	134	7808	127	7825

5.4.2 Experimental results of TPI for reducing large FFRs techniques

The experimental results of the four TPI techniques for reducing large FFRs (TPI_{FFR}), described in the previous subsection, are shown in Table 5.7 [Geu02b]. The compact ATPG test set sizes after TPI are listed for ISCAS circuits and industrial benchmark circuits that have FFRs with more than 200 inputs. The FFR sizes for the ISCAS and industrial benchmark circuits are listed in Appendix A, respectively Appendix B. The experiments have been run on an AMD Athlon 1600+ computer with 512Mb of DDR RAM running RedHat Linux 7.3.

Column $TSFF_{TOT}$ shows for the circuits listed in Column *Circuit* the total number of inserted TPs. Column $T_{No\ FFR}$ shows the compact ATPG test set sizes after TPI, using the Hybrid TPI algorithm with the TC&COP based CF given in Eq. 5.10, but without reducing the large FFRs. Columns *Method A1*, *Method A2*, *Method B1*, and *Method B2* show the compact ATPG test set sizes T after the insertion of $TSFF$ TPs with TPI for reducing large FFR method A1, respectively A2, B1 and B2. Note that *all* listed test set sizes are the sizes after $TSFF_{TOT}$ TPs are inserted. In case of TPI for reducing large FFRs, first $TSFF$ TPs are inserted with methods A1, respectively A2, B1, and B2. The remaining TPs, ($TSFF_{TOT}$ - $TSFF$), are inserted using the Hybrid CRF TPI algorithm.

In total over all circuits listed in Table 5.7, 7699 ATPG patterns are generated after the insertion of 1753 TPs without reducing large FFRs. In case TPI for reducing large FFR Method *A1* is used, 232 of the 1753 TPs are inserted to reduce the large FFRs. With TPI for reducing large FFRs Method *A1*, the number of ATPG test patterns after TPI is reduced from 7699 down to 6592 patterns, which is a reduction of 14.4% ($\frac{7699-6592}{7699} \cdot 100\%$) compared to TPI without reducing large FFRs. In case of TPI for reducing large FFR Methods

A2, B1, and B2, the number of ATPG patterns after TPI is reduced from 7699 to 6654 patterns (13.6% reduction compared to TPI without reducing large FFRs) for Method A2, respectively to 6913 patterns (10.2% reduction) for Method B1, and to 6960 patterns (9.6% reduction) for Method B2. Overall, Method A1 results in the best test set size reduction.

From these results one can conclude that using definition *A* for a large FFR results in better test set size reduction than definition *B*; Methods *A1* and *A2*, which are based on definition *A*, have better test set size reduction than Methods *B1* and *B2*, which are based on definition *B*. These differences in reduction are mainly caused by the large difference in test set size reduction for circuit p73257. This circuit contains over 50 FFRs with more than 200 inputs and because of this, $\overline{FFR} + 3 \cdot \sigma^{FFR}$ is larger than 269 (the largest FFR) and no TPs are inserted by TPI for reducing large FFRs methods *B1* and *B2*. Although this circuit only contains semi-large FFRs, it does suffer from a large test set and the results of TPI for reducing large FFR methods *AI* and *A2* show that TPI for reducing large FFRs results in significantly better test set size reduction.

In contrast to the results of *NPAT* TPI and TC&COP TPI shown in Table 5.5, the results in Table 5.7 show that it is possible to achieve significant compact ATPG test set size reduction for circuit p598004, as long as TPI for reducing large FFRs is used. With TPI for reducing large FFRs, the ATPG test set size for this circuit is reduced to 1320-1345 patterns which is considerably less than the 2129 patterns without TPI.

The experimental results in [Geu02b] showed that far more TPs were inserted with Methods *B* than with Methods *A*. The results in Table 5.7 show that fewer TPs are inserted with Methods *B* than with Methods *A*. Due to a bug in the TPI tool, the $\overline{FFR} + 3 \cdot \sigma^{FFR}$ was not set correctly in [Geu02b]; as a result far too many TPs were inserted with Methods *B* in [Geu02b] for the larger circuits. Inserting more TPs with a TPI for reducing large FFR method does not always result in a better test set size reduction. For circuit p481470 more TPs are inserted with Methods *A* than with Methods *B*, but the test set size reduction for Methods *B* are better than for Methods *A*; i.e., 1428-1468 patterns for Methods *B* versus 1657-1705 patterns for Methods *A*. Too many TPs in circuit p481470 are inserted with the TPI for reducing large FFR Methods *A*, i.e., 111 out of 185, such that only a small number of TPs are inserted with the Hybrid TPI algorithm. More TPs with the Hybrid TPI algorithm are probably required for this circuit to solve the ATPG testability problems in this circuit, other testability problems than large FFRs.

Method *A1* results in better test set size reduction than Method *A2*, with the same number of TPs inserted for reducing large FFRs. Also Method *B1* results in better test set size reduction than Method *B2*. Given these results, it can be concluded that using Strategy *1* to reduce large FFRs results in better test set size reduction than Strategy *2*. More equally sized FFRs seems more important for the ATPG tool than better test counts reduction.

Overall, the results in Table 5.7 show that by using TPI for reducing large FFR methods for circuits with large FFRs, an extra ATPG test set size reduction of 9.6% to 14.4% can be obtained. Method *A1* results in the best test set size reduction. In the remaining part of this dissertation, TPI for reducing large FFR method A1 is used as a TPI pre-process of the Hybrid TPI algorithm.

5.5 Multi-stage TPI with dynamic cost function selection

Every circuit has its own characteristics and therefore also its own testability problems. Some circuits suffer from many RPR faults, while other circuits suffer from signal lines with high TCs, as described in Section 5.3. It is also possible that the circuit suffers from a combination of testability problems. The Hybrid CRF TPI algorithm for industrial circuits uses a CF to determine where in the circuit TPs should be inserted. It is important that a CF is used that takes into account the TA measures for the testability problems that really exist in the circuit. E.g., using a TC based CF, while the circuit does not suffer from high TCs, does not seem to be useful.

In [Geu02b] we proposed a TPI pre-process that selects a CF for the Hybrid TPI algorithm for industrial circuits that tries to solve the hardest test problem that is found in the circuit. The TA measures, extracted from the circuit, are analyzed to identify these test problems. Instead of inserting all TPs using the selected CF, we proposed to split the TPI process into multiple stages. In each stage, the TPI pre-process is run to select the best CF given the TA measures of the current circuit (including already inserted TPs from previous stages). Several TPs are inserted by the TPI algorithm using this CF. In the next stage, the TPI pre-process is run again to select a new cost function.

The CF selection procedure introduced in [Geu02b] is described in Subsection 5.5.1 and the corresponding experimental results are shown in Subsection 5.5.2.

5.5.1 The cost function selection procedure

COP, SCOAP and TC give, for each line, measures for different possible testability problems in the circuit; i.e., COP gives measures for the PR detectability, SCOAP for the number of necessary input assignments and TC for the number of tests that have to propagate through a line. The TPI pre-process analyzes the TA measures to find out whether they indicate that there are testability problems. During the analysis, priorities are assigned on improving the COP, SCOAP and TC measures. The higher the priority for a TA measure, the higher the testability problem corresponding to this TA measure and the more important it becomes to improve these TA measures. For the different TA measures, the following priority schemes are used:

COP priority:

The COP priority is determined by the number of faults with a COP detectability (Pd_f) below a given threshold value. The more faults with a Pd_f below this threshold, the higher the priority. The priority is also higher, when there are faults with very low Pd_fs, e.g., below 1^{-10}, to make sure that the TPI algorithm will target these very hard-to-test faults, even when there are only a few of them.

SCOAP priority:

The SCOAP priority is determined in a similar way as the COP priority. The number of faults with SCOAP values higher than a given threshold determine the priority. As mentioned in Subsection 5.2.1, relative SCOAP values (SCOAP values of a given signal line compared to SCOAP values of other lines) are more useful than absolute SCOAP values. Therefore the threshold is determined by the distribution of the SCOAP values and is set at $\overline{SCOAP} + 3 \cdot \sigma_{SCOAP}$ (standard deviation). This threshold equation has been chosen experimentally.

TC priority:

For the TC only two priority values are used, 0 and 1. Experiments with more priority values have not taken place. Again a given threshold is used to differentiate between these priorities. When there are TC values ($T0$+$T1$) for lines higher than this threshold, the priority is 1, otherwise 0. This threshold has experimentally been set at 40.

When the COP priority is higher than the SCOAP priority, a COP-based CF is selected. On the other hand, when the SCOAP priority is higher, a SCOAP based CF is selected. When COP and SCOAP have an equally high priority, a CF is selected that takes into account both COP and SCOAP. The selected CF will also take into account TCs when the TC priority is 1.

Two kinds of COP CFs have been described in this dissertation, the original COP based CF of Tsai et al. [Tsa97]: $\frac{1}{Pd_f}$, from now on called "Tsai CF", and our proposed NPAT CF: $(1 - Pd_f)^{NPAT}$. The CF selection procedure also has to select which kind will be used in the selected CF. Subsection 4.3.3 already described that the Tsai CF mainly focuses on improving the detection probability of the hardest RPR fault(s), while the $NPAT$ CF focuses on improving the detection probabilities of all faults with low detection probabilities. These faults will have a cost contribution of (nearly) 1. Several properties of the $NPAT$ CF are described in the following text in order to explain how to choose between the Tsai and the $NPAT$ CF and how to select the $NPAT$ value.

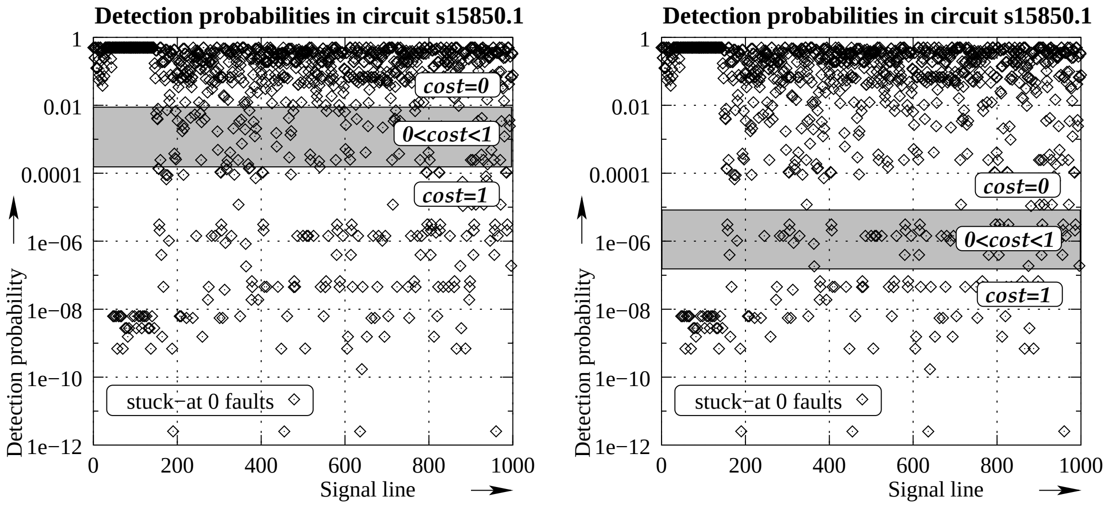
The $NPAT$ CF divides the faults in the circuit into three groups:

Group 1: The faults with a cost contribution of (almost) 1.

Group 2: The faults with a cost contribution of (almost) 0.

Group 3: The faults with a cost contribution larger than 0 but smaller than 1.

Whether a fault belongs to Group 1, 2 or 3 depends both on its detection probability *and* on the value of $NPAT$. This is illustrated in Fig. 5.4. The detection probabilities of the SA0 faults for the first 1000 signal lines in circuit s15850.1 are shown. Fig. 5.4(a) shows the different fault groups in case $NPAT$=320. *COST=1* represents the group with a cost contribution of almost 1 (>0.95), *COST=0* the group with a cost contribution of almost 0 (<0.05) and *0<COST<1* the group with a cost contribution between 0.05 and 0.95.



(a) Cost contributions for *NPAT*=320 (b) Cost contributions for *NPAT*=320000

Figure 5.4: Impact of *NPAT* on the cost contributions of faults in circuit s15850.1

The Hybrid TPI algorithm targets at reducing the global cost of the circuit. The algorithm tries to improve detection probabilities of faults in the circuit that results in a cost contribution reduction from non-zero ($\gg$0) to (almost) 0, hence the TPI algorithm targets the faults in Group 1, and partially the faults in Group 2. But when *NPAT* is low, there are a lot of faults in Groups 1 and 2. Given Fig. 5.4(a) with *NPAT*=320, it is possible that the TPI algorithm insert TPs that improve the detection probabilities of lots of faults with detection probabilities between 10^{-3} and 10^{-4}, while it does not improve the detection probabilities of the really hard-to-test faults, i.e., faults with detection probabilities below 10^{-7}.

When *NPAT* is increased to 320000, shown in Fig. 5.4(b), the faults with detection probabilities between 10^{-3} and 10^{-4} have a cost contribution of (almost) 0 and the TPI algorithm will not target these faults. Now the TPI algorithm will mainly target on the hard-to-test faults, the faults with a detection probability lower than 10^{-7}.

In case the cost selection procedure selects the *NPAT* CF as COP based CF, the *NPAT* parameter will be selected such that approximately 5% of the faults will be in Groups 1 and 2. The TPI algorithm will mainly focus on improving the detection probability of the 5% hardest-to-test faults and will not be influenced by the less hard-to-test faults.

At this point still the question has to be answered when to use the Tsai COP CF and when to use the *NPAT* CF. In case there are very low detection probabilities in the circuit, often the difference in detection probabilities within the 5% lowest detection probabilities is high, i.e., many orders (>5) of magnitude. In that case it is still possible that although a TP might improve the detection probability of a fault with several orders of magnitude, its *NPAT* cost contribution hardly changes and stays almost 1. This is illustrated in Fig.

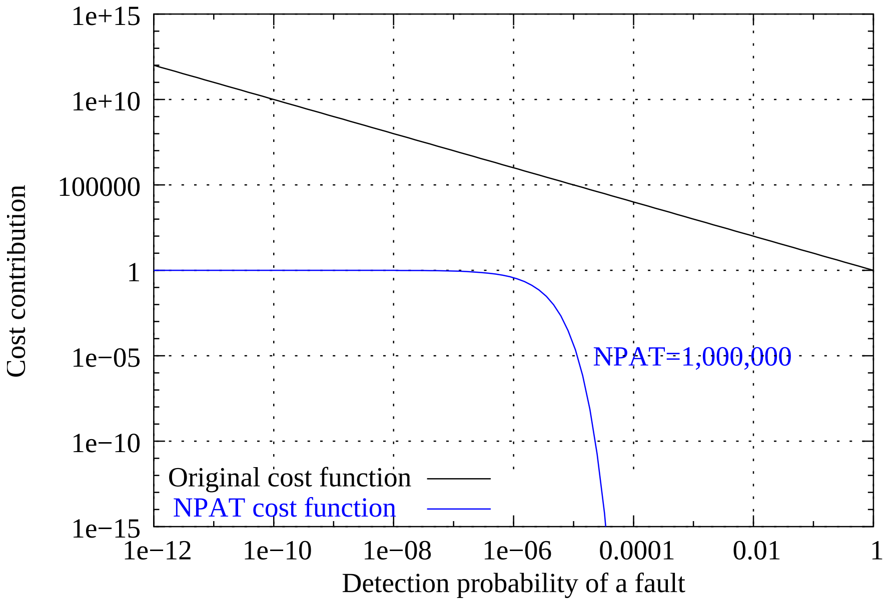


Figure 5.5: Cost contribution of a fault as function of its detection probability

5.5.

In Fig. 5.5 the cost contribution of a fault is plotted as function of its detection probability. The cost contributions are shown for both the Tsai CF and the *NPAT* CF with *NPAT*=1,000,000. Assume that the detection probabilities of the 5% faults with the lowest detection probability ranges from 10^{-12} to 10^{-6}. When a TP candidate improves the detection probability of a fault from 10^{-12} to 10^{-7}, the *NPAT* cost contribution of that fault hardly changes, i.e., remains almost 1. Even when for many faults the detection probability improves from 10^{-12} to 10^{-7}, the total cost for the circuit will hardly change because the cost contribution for each of these faults remains almost 1. The impact of detection probability changes on the cost in this range of detection probabilities is very limited. On the other hand, the Tsai CF results in a cost gain of almost 10^{12} (10^{12} - 10^{7}) for that fault. When for more faults the detection probability is improved from 10^{-12} to 10^{-7}, the cost gain will even be higher. The TPI algorithm with the Tsai CF will consider this TP candidate as possible TP to insert, while the TPI algorithm with *NPAT* CF will certainly not. In case of the *NPAT* CF only TPs are inserted which result in detection probability changes from 10^{-7} and lower to 10^{-6} and higher.

When a higher value of *NPAT* is selected, fewer faults will contribute to the CF and the range of non-zero cost contributing faults becomes smaller. In that case it becomes more likely that a detection probability improvement for a hard-to-test fault will result in a cost contributing change from almost 1 to almost 0, because the range in which the impact of detection probabilities on the cost is limited, has become much smaller. But in that case, the TPI algorithm will only focus on the very hard-to-test faults and ignore all other faults. It will even focus more on the very hard-to-test faults only than the Tsai CF.

Consider again two faults f_1 and f_2, $Pd_{f_1} = 10^{-11}$ and $Pd_{f_1} = 10^{-8}$ and a value for *NPAT* of 1,000,000,000. The cost contribution of fault f_1=0.99, while the cost contribution of fault f_2=$4.5 \cdot 10^{-5}$, a difference of 5 orders of magnitude. The TPI algorithm with *NPAT* CF will ignore fault f_2. In case of the Tsai CF the difference in cost contribution is 'only' 3 orders of magnitude. Still this is not a high probability that the TPI algorithm will take into account cost changes of fault f_2, but it will take them sooner in account than with the *NPAT* CF.

So when does the *NPAT* CF become interesting? The *NPAT* CF becomes interesting when the range in detection probabilities of the faults with non-zero cost contribution is less than 3 orders of magnitude [Geu02a]. Consider again the two faults f_1 and f_2, this time $Pd_{f_2} = 10^{-9}$ which results in a cost contribution of 0.36. This *NPAT* cost contribution is in the same order of magnitude as the contribution of fault f_1, hence the TPI algorithm with NPAT CF will focus on improving the detection probability of both faults. In case of the Tsai CF, the difference in cost contribution for faults f_1 and f_2 is 2 orders in magnitude. Therefore, the TPI algorithm with the Tsai CF will mainly focus only on solving the detection probability of fault f_1.

The CF selection procedure selects the *NPAT* COP based CF when the detection probability range of cost contributing faults is less than 3 orders of magnitude, otherwise the Tsai COP based CF is selected.

The list of CFs from which the CF selection procedure can select is given in Table 5.8. Column # in Table 5.8 shows an identification for the used CF, e.g., CF$_1$ corresponds with the Tsai CF. Column *Cost function* gives the CF equations for the SA1 fault for line l. In these equations, $Pd_{l/SA1}$ represents the detection probability of the SA1 fault at line l, $C0_l$ the COP 0-controllability of line l, $SC0_l$ and SW_l the SCOAP 0-controllability,

Table 5.8: Cost functions for solving different test problems in a circuit*)

#	Cost function	Based on
CF$_1$	$\frac{1}{Pd_{l/SA1}}$	COP
CF$_2$	$(1 - Pd_{l/SA1})^{NPAT}$	COP
CF$_3$	$(SC0_l + SW_l)$	SCOAP
CF$_4$	$(T0_l)$	TC
CF$_5$	$\frac{SC0_l + SW_l}{Pd_{l/SA1}}$	COP & SCOAP
CF$_6$	$(SC0_l + SW_l) \cdot (1 - Pd_{l/SA1})^{NPAT}$	COP & SCOAP
CF$_7$	$\frac{T0_l - E0_l}{C0_l} + \frac{E0}{Pd_{l/SA1}}$	COP & TC
CF$_8$	$(T0_l - E0_l) \cdot (1 - C0_l)^{NPAT} + E0_l \cdot (1 - Pd_{l/SA1})^{NPAT}$	COP & TC
CF$_9$	$(T0_l - E0_l) \cdot (SC0_l) + E0_l \cdot (SC0_l + SW_l)$	SCOAP & TC
CF$_{10}$	$\frac{(T0_l - E0_l) \cdot SC0_l}{C0_l} + \frac{E0_l \cdot (SC0_l + SW_l)}{Pd_{l/SA1}}$	COP & SCOAP & TC
CF$_{11}$	$(T0_l - E0_l) \cdot SC0_l \cdot (1 - C0_l)^{NPAT}$ $+E0_l \cdot (SC0_l + SW_l) \cdot (1 - Pd_{l/SA1})^{NPAT}$	COP & SCOAP & TC

*) Only the stuck-at 1 parts of the cost functions are shown

respectively SCOAP observability of line l, $E0_l$ and $T0_l$ the essential-zeros, respectively total-zeros counts for line l. Only the part of the CF with respect to the SA1 fault is shown in order to limit the size of the table. Finally, Column *Based on* shows on which TA measures this CF is based. Given that there are two possible COP-based CFs, there are 11 possible CFs: two COP-only, one SCOAP-only, one TC-only, two COP&SCOAP, two COP&TC, one SCOAP&TC, and two COP&SCOAP&TC based CFs.

5.5.2 Experimental results of multi-stage TPI

This subsection shows the impact of multi-stage TPI, with the TPI pre-process that selects the CF, on the compact ATPG test set sizes. The results of multi-stage TPI [Geu02b] are compared with the results of the COP&TC based CF proposed in Section 5.3. The comparison results are given in Tables 5.9 and 5.10 for the ISCAS respectively industrial circuits. Multi-stage TPI includes the TPI pre-process for reducing large FFRs [Geu02b] as described in Section 5.4.

Column *TC&COP TPI* shows the compact ATPG results for the circuits listed in Column *Circuit* after TC&COP TPI; the ATPG test set size (*T*), the ATPG fault coverage (*FC(%)*) and the CPU time spent on ATPG (*CPU*) are listed. Column *TPI* shows the number of inserted TSFF TPs (*TSFF*) and the CPU time spent on multi-stage TPI (*CPU*). Column *multi-stage TPI* shows the compact ATPG results for the circuit when multi-stage TPI has been used; the ATPG test set size (*T*), the ATPG fault coverage (*FC(%)*) and the CPU time spent on ATPG (*CPU*) after multi-stage TPI are listed.

The compact ATPG results, listed in Table 5.9, show that with multi-stage TPI 48 fewer ATPG patterns are generated than with TC&COP TPI, i.e., 1014 patterns are generated after multi-stage TPI versus 1062 patterns after TC&COP TPI. In other words, multi-stage TPI results in 4.5% extra test set size reduction compared to TC&COP TPI. For all three sets of ISCAS circuits, i.e., ISCAS'85, ISCAS'89 and ISCAS'89 addendum, better test set size reduction is achieved with multi-stage TPI. However, there are three circuits for which the test set size reduction is not as good as with TC&COP TPI, i.e., circuits s9234.1, s13207.1 and s938a. For these circuits, the multi-stage TPI does not select the best CF, although still very good test set size reduction has been achieved after TPI compared to the results without TPI, see Appendix A. On the other hand, the number of compact ATPG test patterns for circuits c7552, s15850.1 and s38584.1 is considerably smaller with multi-stage TPI compared to TC&COP TP. For these circuits it is clearly shown that the multi-stage TPI results in better ATPG testability improvement.

The ATPG fault coverages of the ISCAS circuits after TC&COP TPI and multi-stage TPI do not differ a lot, i.e., 0.01% (99.02%-99.01%) for the ISCAS'85 circuits, 0.16% for the ISCAS'89 circuits, and 0.13% over all ISCAS circuits. For all sets of ISCAS circuits, multi-stage TPI results in the better fault coverage.

Also the ATPG CPU times do not differ a lot. Overall, the ATPG CPU times for the

Table 5.9: Compact ATPG results of ISCAS circuits after multi-stage TPI

	TC&COP TPI			TPI		multi-stage TPI		
Circuit	T	FC(%)	CPU	TSFF	CPU	T	FC(%)	CPU
c1908	42	99.92	0.46 s	25	1.25 s	40	99.92	0.45 s
c3540	73	97.67	1.69 s	25	4.45 s	70	97.73	2.26 s
c7552	74	99.40	3.40 s	40	7.58 s	56	99.40	3.01 s
Subtotal	189	99.01	5.55 s	90	13.3 s	166	99.02	5.73 s
s1196	52	100.00	0.30 s	20	0.65 s	52	100.00	0.31 s
s1488	88	100.00	0.44 s	20	0.91 s	81	100.00	0.47 s
s1494	87	99.87	0.46 s	20	0.92 s	82	99.93	0.45 s
s5378	60	98.91	1.53 s	30	2.60 s	58	98.98	1.62 s
s9234.1	51	94.69	4.24 s	50	8.49 s	57	94.60	4.65 s
s13207.1	202	99.16	11.0 s	50	12.8 s	212	99.15	11.5 s
s15850.1	88	98.38	9.72 s	50	13.2 s	75	98.36	8.75 s
s38584.1	110	96.53	36.4 s	50	34.9 s	98	96.91	34.4 s
Subtotal	738	97.39	64.0 s	290	74.4 s	715	97.55	62.2 s
s499a	28	100.00	0.18 s	8	0.21 s	28	100.00	0.18 s
s938a	40	100.00	0.25 s	12	0.44 s	41	100.00	0.25 s
s3330a	67	100.00	1.03 s	32	1.89 s	64	100.00	1.06 s
Subtotal	135	100.00	1.46 s	52	2.54 s	133	100.00	1.48 s
Total	1062	97.71	71.1 s	432	90.2 s	1014	97.84	69.4 s

ISCAS circuits after multi-stage TPI are 1.7 seconds less than after TC&COP TPI (69.4 seconds for multi-stage TPI versus 71.1 seconds for TC&COP TPI).

The results of the industrial benchmark circuits, listed in Table 5.10, also show that multi-stage TPI results in significantly better compact ATPG test set size reduction than TC&COP TPI. For the Boolean industrial circuits, the ATPG test set size is reduced from 4200 patterns with TC&COP TPI to 3050 patterns with multi-stage TPI. This means that with multi-stage TPI a test set size reduction of $\frac{4200-3050}{4200} \cdot 100\% = 27.4\%$ has been achieved compared to TC&COP TPI. Especially for circuits p73257 and p162057, better test set size reduction has been achieved. In case of circuit p162057, this better reduction is totally the result of the usage of the TPI for reducing large FFRs pre-process in multi-stage TPI, see Table 5.7. In case of circuit p73257, this is only partially true. After multi-stage TPI on circuit p73257, the ATPG test set size contains 413 patterns less (566 patterns versus 979 patterns) than with TPI for reducing large FFR method *A1* in combination with TC&COP TPI, see Table 5.7. Compared to TC&COP TPI without reducing large FFRs, multi-stage TPI on circuit p73257 results even in 56.7% ($\frac{1308-566}{1308} \cdot 100\%$) extra test set size reduction.

The fault coverage differences between TC&COP TPI and multi-stage TPI for the

Table 5.10: Compact ATPG results of industrial circuits after multi-stage TPI

Circuit	TC&COP TPI			TPI		multi-stage TPI		
	T	FC(%)	CPU	TSFF	CPU	T	FC(%)	CPU
p7653	178	99.62	8.74 s	15	6.91 s	158	99.63	8.66 s
p14148	260	99.99	19.9 s	15	5.68 s	212	99.99	17.3 s
p27811	97	95.22	25.2 s	38	16.0 s	87	95.22	25.6 s
p31025	463	98.87	129 s	25	30.9 s	445	98.74	122 s
p34592	198	99.98	75.4 s	34	22.1 s	173	99.97	70.1 s
p36503	159	97.25	48.1 s	30	23.5 s	125	97.25	42.2 s
p43282	197	97.73	127 s	49	49.9 s	183	97.79	118 s
p43663	132	99.25	40.1 s	50	28.5 s	115	99.26	38.9 s
p72767	302	97.25	404 s	50	147 s	304	97.41	400 s
p73133	285	97.39	404 s	50	164 s	270	97.30	393 s
p73257	1308	98.09	561 s	73	133 s	566	98.25	356 s
p75344	174	99.31	102 s	55	70.4 s	182	99.33	109 s
p162057	447	98.69	348 s	162	627 s	230	98.79	273 s
Subtotal	4200	98.28	2296 s	646	1326 s	3050	98.32	1977 s
p32118	147	93.56	30.2 s	32	20.0 s	134	93.57	30.7 s
p37021	176	77.28	23.2 s	37	25.3 s	208	77.23	24.0 s
p66171	1177	96.79	354 s	66	379 s	1138	96.79	348 s
p71553	96	96.86	290 s	71	145 s	87	96.78	228 s
p93140	325	94.03	587 s	93	173 s	386	94.03	614 s
p104649	476	99.39	367 s	104	191 s	480	99.39	346 s
p114605	366	99.01	500 s	80	173 s	381	98.94	526 s
p137498	274	98.86	177 s	137	308 s	298	98.85	200 s
p481470	1435	99.28	8390 s	185	904 s	1693	99.26	11283 s
p596922	521	96.97	2595 s	317	2894 s	450	96.95	2953 s
p598004	2069	98.74	4313 s	100	932 s	1440	98.74	4022 s
p705050	286	76.86	6817 s	250	2653 s	256	76.86	6568 s
p824184	759	96.73	18655 s	300	3952 s	749	96.74	18123 s
p854266	411	97.65	5442 s	300	3911 s	414	97.68	6136 s
Subtotal	8518	94.42	48546 s	2072	16666 s	8114	94.42	51406 s
Total	12718	94.92	50842 s	2718	17992 s	11164	94.92	53384 s

Boolean industrial circuits are very limited, 98.28% after TC&COP TPI versus 98.32% after multi-stage TPI. For seven of the thirteen Boolean industrial circuits, multi-stage TPI results in the better fault coverage, while TC&COP TPI only results three times in the better fault coverage.

The compact ATPG CPU times after multi-stage TPI are also less than the compact ATPG CPU times after TC&COP TPI; 1977 seconds versus 2296 seconds. In other words, multi-stage TPI results in $\frac{2296-1977}{2296} \cdot 100\% = 13.9\%$ extra ATPG CPU time reduction compared to TC&COP TPI. Again this difference is mainly caused by circuits p73257 and p162057. For the ATPG tool it becomes much easier to activate faults or propagate fault effects when there are fewer large FFRs with high TCs.

The compact ATPG test set size reduction after multi-stage TPI for the three-state industrial circuits is less compared to the reduction for the Boolean industrial circuits. 8114 ATPG patterns are generated after multi-stage TPI compared to 8518 patterns after TC&COP TPI; in other words a reduction of $\frac{8518-8114}{8518} \cdot 100\% = 4.7\%$. A very large amount of this difference is caused by the usage of TPI for reducing large FFRs within multi-stage TPI, see the results of circuits p481470 and p598004 in Tables 5.7 and 5.10. The multi-stage TPI algorithm is not able to find really better CFs for the three-state industrial circuits compared to the TC&COP CF, i.e., CF_7. The larger three-state industrial circuits mainly suffer from poor COP detectability and high TC values, which is exactly what TC&COP TPI tries to improve.

The ATPG fault coverages after multi-stage TPI and TC&COP TPI are the same, i.e., 94.42%. Just like TC&COP TPI, multi-stage TPI is also not able to improve the fault coverage for circuit p705050, which *NPAT* TPI was able to do (see Tables 5.2 and 5.5). For all three-state industrial circuits, the differences in ATPG fault coverages after TPI is limited to less than 0.1%.

Multi-stage TPI results in longer ATPG CPU times than TC&COP TPI for the three-state industrial circuits. This is mainly caused by circuit p481470. As explained in Subsection 5.4.2, most of the inserted TPs in circuit p481470 are inserted by the TPI for reducing large FFR method *A1* (included in multi-stage TPI), i.e., 111 out of 185 TPs. Only a relatively small number of TPs is inserted by the Hybrid CRF TPI method. This circuit requires more TPs to be inserted with the Hybrid CRF TPI method in order to improve other ATPG testability problems in the circuit, i.e., improve COP detectability and/or SCOAP observability.

Overall, the compact ATPG test set size reduction for the industrial benchmark circuits is 12.2% better after multi-stage TPI compared to TC&COP TPI, i.e., 11164 patterns versus 12718 patterns. There is no significant difference in ATPG fault coverage between multi-stage TPI and TC&COP TP; both result in 94.92% ATPG fault coverage. The CPU time spent on ATPG is 5% more for multi-stage TPI compared to TC&COP TP TPI, i.e., 53384 seconds versus 48546 seconds. As explained above, this is mainly due to the many TPs that are inserted in circuit p481470 for reducing large FFRs.

5.6 Summary and conclusions

The growth in ATPG test set sizes, caused by the increasing complexity of circuits, has a major impact on the ATE demands for the semiconductor industry. In this chapter it has been shown that TPI can be used to reduce the compact ATPG test set sizes, hence reduce the ATE memory requirements and the long test time needed to apply the generated ATPG test sets. The TPI algorithm with the *NPAT* CF, which is only targeted on improving the PR testability of a circuit, is capable of significantly reducing compact ATPG test set sizes.

For the set of ISCAS benchmark circuits, 43.5% test set size reduction has been achieved, and for the industrial benchmark circuits, the test set size reduction was 26.1%. Besides reducing the ATPG test set sizes, *NPAT* TPI also resulted in an improvement of the ATPG fault coverage and a reduction of the CPU time spent on ATPG. For the ISCAS circuits, the ATPG fault coverage improved from 97% to 98%; this means that 33% of all faults that were not covered by ATPG patterns before TPI, became detectable after *NPAT* TPI. The ATPG CPU time spent on all ISCAS benchmark circuits has been reduced with 31.8%.

For the industrial benchmark circuits, the ATPG fault coverage improved from 94.76% to 96.78%; in other words 40% of the faults that were not covered by ATPG patterns without TPI, became covered after *NPAT* TPI. The ATPG CPU time spent on the industrial circuits has been reduced with 65.1%.

Although results show that, in general, significant test set size reductions can be achieved with *NPAT* TPI, there are however circuits for which the compact ATPG test set size reduction was not so high; i.e., less than 10%. These circuits suffer from other testability problems that cause large test sets than only poor PR testability. We have proposed [Geu00] a CF for the Hybrid TPI algorithm for industrial circuits that not only takes into account COP measures, but also test counts (TCs) [Hay74]. With TCs a lower bound on the compact ATPG test set size can be calculated. High TC measures indicate that the ATPG test set size will also be high. Reducing the TCs may result in a reduction of the ATPG test set size.

Experimental results on both the ISCAS benchmark and industrial circuits have shown that the proposed TPI algorithm with the CF that takes TCs into account, i.e., TC&COP TPI, in general results in a better test set size reduction than *NPAT* TPI. Although for the ISCAS benchmark circuits, only 1.7% extra test set size reduction is achieved, for the industrial circuits 17% extra test set size reduction is achieved after TC&COP TPI compared to *NPAT* TPI. The ISCAS circuits do not contain circuits that suffer from high TC values, while several industrial circuits do; this is the cause for the difference in extra test set size reduction between the ISCAS circuits and the industrial circuits.

Even with TC&COP TPI, there are still circuits for which the test set size reduction was disappointing, e.g., for circuit p598004 less than 3% reduction had been achieved after the insertion of 100 TPs with TC&COP TPI compared to the circuit without TPs. This circuit suffers from a very large FFR that results in very high TC values on the FFR output. Still, the TC&COP TPI (or *NPAT* TPI) algorithm did not insert TPs in this large FFR. By extending the TPI algorithm with methods to reduce the sizes of large FFRs, also significant test set size reduction can be achieved for circuits that suffer from very large

FFRs. Experimental results have shown that with these methods, also the test set size for circuit p598004 can be reduced with 38%, from 2129 patterns down to 1320. Over the set of industrial benchmark circuits that contain large FFRs, 9.6% up to 14.4% extra test set size reduction has been achieved when the TPI algorithm includes a TPI pre-process that reduces the large FFRs by means of one of the TPI for reducing large FFR methods.

The ISCAS circuits do not suffer from high TCs; using a TC-based CF like in TC&COP TPI does not seem useful for these circuits. In general each circuit suffers from different testability problems. In this chapter we have proposed to use a TPI pre-process, that analyzes the TA measures of the circuit, to select a CF aimed at solving the testability problems of that specific circuit. The TPI process is divided into multiple-stages. Before each stage, the TPI pre-process is run to select the CF that targets the worst testability problems in the circuit. Several TPs are inserted with that CF, after which the pre-process is run again to select a new CF. Because the inserted TPs improve the testability problems in the circuit, another CF may become better to target the testability problems that exist after the currently inserted TPs. In this chapter we have shown that by dividing the TPI process into multiple stages, in order to select in each state a CF after an analysis of the TA measures, even better test set size reductions are achieved than using a COP-only (i.e., *NPAT*) CF or the TC&COP CF during the whole TPI process.

Experimental results of the proposed multi-stage TPI method, i.e., multi-stage TPI, show that for the ISCAS circuits the number of ATPG patterns is reduced with 4.5% compared to TC&COP TPI. For the industrial circuits, the extra reduction is 12.2% compared to TC&COP TPI. Especially the reduction for the Boolean industrial circuits is large, i.e., 27.4%. The reduction for the three-state industrial circuits is 4.7%.

Comparing the compact ATPG results of multi-stage TPI with the ATPG results of the circuits without TPs, for the ISCAS circuits the number of ATPG test patterns is reduced from 1913 down to 1014 patterns; a reduction of 47%. The ATPG fault coverage is improved from 97% to 97.84% and the CPU time spent on ATPG is reduced from 111 seconds down to 69.7 seconds. For the industrial benchmark circuits, the number of ATPG patterns is reduced from 22987 patterns to 11164 patterns; a reduction of 51.4%. The ATPG fault coverage has only been improved from 94.76% to 94.92%. For one circuit, i.e., p705050, *NPAT* TPI was able to improve the fault coverage from 76.76% to 91.47%, resulting in a fault coverage improvement over all industrial circuits of 2%. TC&COP TPI and multi-stage TPI were not able to improve the fault coverage for this circuit significantly, resulting in only a small overall improvement in the ATPG fault coverage. The CPU time spent on ATPG is reduced from 186185 seconds without TPs to 52758 seconds after multi-stage TPI, a reduction of even 71.6%. Hence, multi-stage TPI can significantly facilitate compact ATPG in reducing the number of patterns to reach a high fault coverage as well as reduce the CPU time that the ATPG requires for generating the patterns.

CHAPTER 6

Test Point Insertion for delay fault ATPG

Besides the impact of TPI on *stuck-at fault (SAF)* ATPG test set sizes, we have also tested the impact of TPI on *gate-delay fault (GDF)* ATPG test set sizes. The compact GDF ATPG test sets tend to be even larger than the compact SAF ATPG test sets, as will be explained in Section 6.1, and hence also form a big problem for the ATE. It would be nice if TPI could also result in significant delay fault ATPG test set size reductions.

Section 6.1 starts with describing the *Transition Fault (TF)* model, a special case of the gate-delay fault model. Section 6.1 also briefly describes GDF ATPG (TF ATPG), including the possible impact of TPI on GDF ATPG. Section 6.2 shows experimental results of TPI for SAF ATPG, described in the previous section, on GDF ATPG. Section 6.3 summarizes and concludes this chapter.

6.1 Transition faults and gate-delay fault ATPG

Chapter 1 has shown that delay faults are often divided into two groups: path-delay faults and gate-delay faults. A special case of the gate-delay faults exists, i.e., the gross gate-delay faults::

gross gate-delay fault [Pra95]: A gate is defined faulty if its gate defect results in path-delay faults for all the paths passing through it.

The gross gate-delay faults can be modeled by the *Transition Fault (TF) model*. For the two kinds of transitions, i.e., the up-transition (the value on a line changes from a 0 to a 1) and down-transition (the value on a line changes from a 1 to a 0), there are two types of TFs:

1. The slow-to-rise fault: With respect to the clock-frequency, the delay of a gate (output) to change from 0 into 1 is too high.

2. The slow-to-fall fault: With respect to the clock-frequency, the delay of a gate (output) to change from 1 into 0 is too high.

The TF model is similar to the SAF-model; the slow-to-rise (slow-to-fall) TF corresponds to a SA0(SA1) fault, since it behaves as a SA0(SA1) fault for a temporary period of time. Given this relationship between the TF-model and the SAF-model, the TF test problem is similar to a two time frame sequential SAF test problem, illustrated by Fig. 6.1. A *time frame* is the state of the circuit, i.e., the state of the FFs, between two clock pulses.

Fig. 6.1 shows the circuit for the two time frames: the *Initial time frame (IT)* and the *Final time frame (FT)*. IT_{PI} (IT_{PO}) and FT_{PI} (FT_{PO}) represent the PI values (PO values) of the circuit in the Initial time frame, respectively Final time frame. IT_{PPI} represents the pseudo-PI values (i.e., SFF output values) in the Initial time frame, and FT_{PPO} represents the pseudo-PO values (i.e., SFF input values) in the Final time frame. The values of the pseudo-PIs (SFF outputs) in the Final time frame, i.e., FT_{PPI}, will be the same as the values of the pseudo-POs (SFF inputs) in the Initial time frame, i.e., FT_{PPO}; after a clock-pulse, the values on SFF inputs are clocked into the FFs.

In the Initial time frame a test pattern is generated to initialize the required transition at the faulty line. In the Final time frame, a stuck-at test pattern is generated to provoke the required transition at the faulty line and to sensitize a path to a circuit output. Both time frames are connected through FFs, i.e., after a clock pulse, the pseudo-PI values in the Final time frame will follow the pseudo-PO values in the Initial time frame. Hence, during TF ATPG, two test generation targets need to be specified: A SAF in the Final time frame and an I_{DDq} modeled fault in the Initial time frame. Just like for an I_{DDq} fault, the fault only needs to be justified in the Initial time frame without being propagated, therefore the term I_{DDq} fault is used.

As a result, when the circuit contains hard-to-control and hard-to-observe lines, one

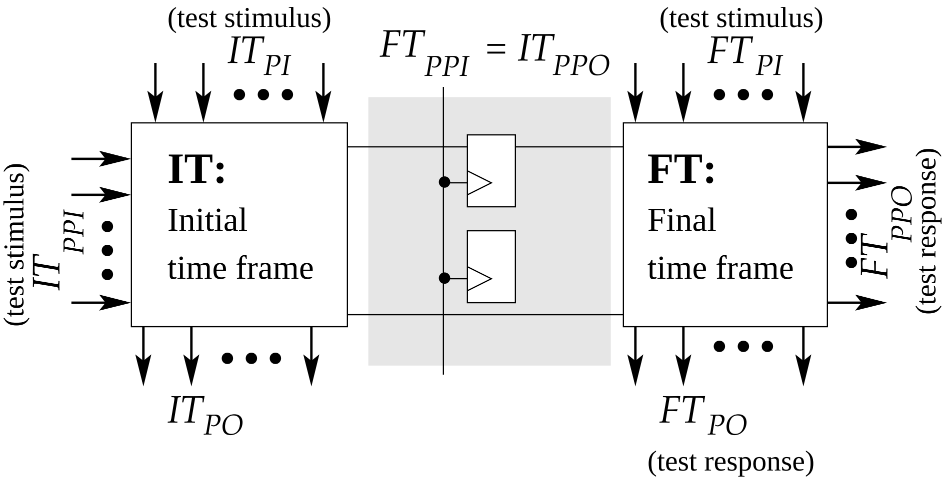


Figure 6.1: Two time frame model for TF ATPG

can imagine that it becomes even harder for the TF ATPG to generate test patterns than for the SAF ATPG. The hard-to-control/hard-to-observe lines exist in both time frames. A TP will make nodes easier controllable/observable in both time frames, which should result in significant TF test set size and CPU time reduction.

6.2 Experimental results of TPI for SAF ATPG on gate-delay fault ATPG

This section gives experimental results of TPI for SAF ATPG on GDF ATPG, i.e., TF ATPG. The TPI algorithm that has been used is the multi-stage Hybrid TPI algorithm described in Section 5.5 [Geu02b]. The experiments have been run on a 1600+ AMD Athlon with 512Mb of DDR RAM, running RedHat Linux 7.3

Subsection 6.2.1 shows the impact of TPI for SAF ATPG on GDF ATPG in general. In this subsection it will be shown that TPI results in GDF test set size reduction, GDF fault coverage/fault efficiency improvement and GDF CPU time reduction. As will be explained, improving the GDF fault coverage/efficiency can result in more GDF test patterns, in other words a negative impact on the GDF test set size reduction. In Subsection 6.2.2 experimental results of TPI for SAF ATPG on GDF test set sizes are shown in case we limit the fault efficiency after TPI to the fault efficiency level for the circuits without TPs, such that we minimize the negative impact of the fault coverage/efficiency improvement on the GDF test set sizes.

6.2.1 The impact of TPI for SAF ATPG on GDF ATPG in general

Tables 6.1 and 6.2 show experimental results of TPI for SAF ATPG on GDF ATPG for the ISCAS, respectively industrial benchmark circuits [Geu02b]. Before TPI has been run, compact GDF ATPG has been run to get the GDF ATPG test set sizes for the circuits without TPs. These GDF ATPG test sizes are listed in Tables A.3 and A.4 of Appendix A for the ISCAS circuits, and in Tables B.3 and B.4 of Appendix B for the industrial circuits. Like for TPI for SAF ATPG, we only apply TPI on the circuits that have a compact GDF test set size of over 100 patterns. Circuits with smaller ATPG test set sizes are not candidate for TPI, because we consider their test set size to be small enough. Due to the very large CPU times of GDF ATPG on larger circuits, and the 512Mb memory limitation of the computer on which the experiments are run, we have limited the experiments to circuits with less than 200,000 signal lines.

In Tables 6.1 and 6.2, Column *Circuit* shows the names of the circuits. Column *GDF ATPG without TPs* shows the compact GDF ATPG results for the circuits without TPs; the GDF test set size (*T*), the GDF fault coverage (*FC(%)*), the GDF fault efficiency (*FE(%)*), and the CPU time spent on GDF ATPG (*CPU*) are listed. Column *TPI* (*TSFF*) shows the data with respect to multi-stage TPI, i.e., the number of TSFFs that have been inserted. Column *GDF ATPG after TPI* shows the compact GDF ATPG results for the circuits after

Table 6.1: GDF ATPG results of ISCAS circuits using multi-stage TPI

Circuit	GDF ATPG without TPs				TPI	GDF ATPG after TPI			
	T	FC(%)	FE(%)	CPU	TSFF	T	FC(%)	FE(%)	CPU
c1355	142	96.22	96.53	2.08 s	16	78	97.65	97.65	0.98 s
c1908	157	96.83	97.07	2.43 s	25	113	97.72	97.82	1.62 s
c3540	162	92.29	95.74	5.89 s	25	126	94.88	97.21	4.56 s
c7552	131	96.67	97.83	11.3 s	40	80	96.89	97.53	8.44 s
Subtotal	592	95.57	97.09	21.7 s	106	397	96.58	97.50	15.6 s
s1196	156	95.64	95.73	1.33 s	20	77	95.72	95.77	0.79 s
s1488	139	89.53	97.08	5.18 s	20	118	95.73	95.77	1.44 s
s1494	143	87.94	96.01	4.79 s	20	127	95.54	95.65	1.57 s
s5378	141	88.54	96.97	7.97 s	30	92	92.15	97.18	4.68 s
s9234.1	232	81.46	92.86	52.3 s	50	99	87.82	95.56	13.1 s
s13207.1	357	83.85	96.39	56.0 s	50	257	88.83	96.94	33.2 s
s15850.1	207	81.43	96.30	59.0 s	50	161	87.12	96.84	36.7 s
s38417	154	94.17	96.12	125 s	50	118	94.16	96.10	84.3 s
s38584.1	229	89.23	96.90	264 s	50	187	91.18	97.51	175 s
Subtotal	1758	88.63	96.21	576 s	340	1236	91.27	96.71	351 s
s938a	184	78.59	96.09	1.36 s	12	73	77.53	94.51	0.62 s
s3330a	151	80.50	90.90	7.40 s	32	96	86.77	96.40	2.95 s
Subtotal	335	80.08	92.04	8.76 s	44	169	84.74	95.98	3.57 s
Total	2685	89.07	96.17	607 s	490	1802	91.61	96.77	370 s

multi-stage TPI; the GDF test set size (*T*), the GDF fault coverage (*FC(%)*), the GDF fault efficiency (*FE(%)*), and the CPU time spent on GDF ATPG (*CPU*) after TPI are listed.

The results in Table 6.1 show that TPI for SAF ATPG also results in significant GDF ATPG test set size reduction. Over all circuits listed in Table 6.1, the number of GDF patterns is reduced from 2685 patterns without TPI to 1802 patterns after multi-stage TPI; a reduction of 32.9% ($\frac{2685-1802}{2685}\cdot 100\% = 32.9\%$). The individual circuit reductions differ a lot, from 11.2% for circuit s1488 (from 139 to 118 patterns) up to 60.3% for circuit s938a (from 184 patterns to 73 patterns). In other words, how successful TPI is on reducing GDF test set sizes, is circuit dependent.

Analyzing the GDF fault coverages before and after multi-stage TPI shows that they significantly improve after TPI for a large number of circuits, i.e., the GDF fault coverage improvements for circuits s1488, s1494, s9234.1, s15850.1 and s3330a are more than 5%. Overall, the fault coverage improvement is 2.54%, from 89.07% to 91.61%. In other words, 23.2% ($(1-\frac{(100\%-91.61\%)}{(100\%-89.07\%)})\cdot 100\% = 23.2\%$) of all GDF faults that were not covered by a GDF pattern for the circuits without TPs, becomes covered by GDF patterns after TPI. This improvement in fault coverage is much larger than the fault coverage improvements seen for the ISCAS circuits in case of SAF ATPG. Tables 5.1 and 5.9 show that the SAF fault coverage after multi-stage TPI only improves from 97.00% to 97.84%. The fault efficiency improvement after TPI is less than the fault coverage improvement; the fault efficiency improves from 96.17% without TPs to 96.77% after TPI, an improvement of 0.6%.

Such a GDF fault coverage improvement means that a lot of GDFs that were undetectable without TPs have become detectable after TPI. For these GDFs also patterns have to be generated, and as a result, the compact GDF ATPG test sizes increase. This makes clear why the GDF ATPG test set size reduction is less than the SAF ATPG test set size reduction. The GDF ATPG has to generate more new patterns for the GDFs that have become detectable, than the SAF ATPG has to generate for the SAFs that have become detectable after TPI as more GDFs than SAFs have become detectable after TPI.

The GDF ATPG CPU times for the ISCAS circuits significantly reduce after TPI; they reduce from 607 seconds without TPs to 370 seconds after TPI. This means a reduction of 39.0%. As explained in Section 6.1, GDF ATPG is more difficult than SAF ATPG; the GDF ATPG has to deal with two time frames while the SAF ATPG only has to deal with one time frame. Also the GDF ATPG version in AMSAL that has been used for GDF ATPG, see Appendix E, is not as far in development as the SAF ATPG and as a result not yet as good in performance. Therefore the GDF CPU times are much larger than the SAF CPU times shown in Tables 5.1 and 5.9.

The results in Table 6.2 shows that TPI for SAF ATPG also results in significant GDF ATPG test set size reductions for the industrial circuits. For the Boolean circuits, the number of GDF patterns reduces with 34.0%, from 15,827 patterns without TPs to 10,452 patterns after multi-stage TPI. Especially circuits p73257 and p162057 have very large GDF test set sizes without TPI. These are two circuits with large FFRs. Like for SAF ATPG, also for GDF ATPG large FFRs form a problem; all fault effects have to propagate through the single FFR output. The used multi-stage TPI algorithm includes TPI for reducing large FFR method *A1*, see Section 5.4, such that the large FFRs will be reduced after TPI. For circuit p162057, reducing the FFR size results in a GDF test set size reduction of 72.4%, from 3578 patterns to 987 patterns. This reduction is far more than the reductions for the other Boolean industrial circuits. The reduction of the GDF patterns for circuit p73257 is not so high. Although the number of GDF patterns is reduced with more than 1000 patterns (from 5697 patterns without TPs to 4612 patterns after multi-stage TPI), the reduction is only 19.0%. This circuit does suffer from large FFRs, but also from other testability problems as explained in Subsection 5.5.2. The Multi-stage TPI is able to reduce the GDF test set sizes for circuit p73257, but not as well as for SAF ATPG, see Table 5.10.

Like for the ISCAS circuits, there is a significant GDF fault coverage improvement of 2.04%, from 90.50% to 92.54%. The fault efficiency improves with 1.09%, from 95.04% to 96.13%. For the Boolean industrial circuits, $\left(1-\frac{(100\%-92.64\%)}{(100\%-90.50\%)}\right)\cdot 100\% = 21.5\%$ of all GDF faults that were not covered by a GDF pattern without TPs, becomes covered by GDF patterns after TPI.

With multi-stage TPI, the CPU time spent on GDF ATPG for the Boolean industrial circuits is reduced from 39,551 seconds without TPs to 17,448 seconds. This is a reduction of 55.9%. Especially the reduction for circuit p162057 is large; the GDF ATPG CPU

Table 6.2: GDF ATPG results of industrial circuits using multi-stage TPI

	GDF ATPG without TPs				TPI	GDF ATPG after TPI			
Circuit	T	FC(%)	FE(%)	CPU	TSFF	T	FC(%)	FE(%)	CPU
p7653	660	90.98	95.04	107 s	15	465	93.24	96.23	52.9 s
p13138	134	95.35	95.88	22.8 s	20	110	96.10	96.88	16.1 s
p14148	623	90.50	94.76	155 s	15	384	89.66	94.43	116 s
p27811	273	82.46	96.64	301 s	38	151	83.47	96.69	190 s
p31025	1185	94.19	96.37	1343 s	25	1014	95.06	96.70	941 s
p34592	310	98.48	98.50	320 s	34	268	99.13	99.14	239 s
p36503	615	83.68	91.31	647 s	30	431	88.10	93.16	327 s
p43282	747	92.60	96.39	1078 s	49	580	93.63	97.16	815 s
p43663	296	94.24	99.27	191 s	50	238	94.69	99.23	157 s
p72767	603	93.02	96.91	5036 s	50	439	93.80	97.42	2019 s
p73133	629	93.29	97.11	3368 s	50	431	94.01	97.61	2012 s
p73257	5697	87.21	89.08	7389 s	73	4612	87.37	89.07	5725 s
p75344	477	91.92	96.99	1156 s	55	342	93.05	96.98	824 s
p162057	3578	87.47	93.06	18431 s	162	987	93.13	96.45	4008 s
Subtotal	15827	90.50	95.04	39551 s	666	10452	92.54	96.13	17448 s
p32118	1189	82.50	97.01	915 s	32	592	85.40	98.99	440 s
p37021	358	38.87	98.43	185 s	37	240	44.27	99.24	132 s
p66171	1067	37.44	41.09	129635 s	66	2777	85.55	89.19	129672 s
p71553	220	96.67	99.91	1497 s	71	169	96.75	99.90	1193 s
p93140	668	93.88	99.17	3450 s	93	499	94.08	99.30	2323 s
p104649	1956	89.60	94.77	11975 s	104	1636	94.25	98.28	6453 s
p114605	2332	95.03	98.52	16641 s	80	1743	95.76	99.11	7758 s
p137498	1082	92.09	98.73	4802 s	137	710	93.22	99.11	2554 s
Subtotal	8872	83.98	92.34	169104 s	620	8366	90.42	98.09	150528 s
Total	24699	87.37	93.74	208655 s	1286	18818	91.52	97.07	167976 s

time for circuit p162057 is reduced with 78.3%, from 18,431 seconds without TPs to 4008 seconds after TPI. For the GDF ATPG tool it becomes much easier to activate faults and propagate fault effects when there are fewer large FFRs.

The reduction of the number of GDF test patterns for the three-state industrial circuits is not as large as the reductions seen for the ISCAS, respectively Boolean industrial circuits. For the three-state industrial circuits, the number of GDF patterns after TPI is reduced with 5.7%, from 8872 patterns to 8366 patterns. This poor reduction is caused by the results of circuit p66171. The number of GDF patterns for circuit p66171 do not decrease, the contrary, they were increased by 160% from 1067 GDF patterns to 2777 GDF patterns. Not taking into account this circuit, the number of GDF patterns for the industrial circuits would reduce from 7805 (8872-1067) patterns to 5589 (8366-2777) patterns, a reduction of 28.4%. 28.4% is more conform the results shown for the ISCAS and Boolean industrial circuits.

But why do the number of GDF patterns increase so much for circuit p66171? During all compact ATPG experiments, both SAF and GDF, the ATPG tools were not allowed to spent more than approximately 36 hours on each circuit. For circuit p66171, the GDF ATPG was only able to generate 1067 GDF patterns for 41.09% of all detectable faults within these 36 hours. After TPI, the GDF ATPG tool was able to generate within 36 hours 2777 GDF patterns for 89.19% of all detectable faults. In addition to the fact that the number of GDF patterns increased enormously after TPI, so did the fault coverage. Without the GDF ATPG time limit, a lot more GDF patterns would be generated, resulting in a higher fault coverage. The results of circuit p66171 show that multi-stage TPI makes it a lot easier for the GDF ATPG tool to generate GDF patterns. Within the same amount of time, i.e., 36 hours, 89.19% of all detectable faults are covered after TPI while only 41.09% of all detecable faults are covered without TPs.

The results of circuit p66171 also show an enormous GDF fault coverage and fault efficiency improvement for the three-state industrial circuits after TPI. The GDF fault coverage improves from 83.98% without TPs to 90.42% after TPI, and the GDF fault efficiency improves from 92.34% to 98.09%. Not taking into account circuit p66171, the GDF fault coverage would improve from 89.20% without TPs to 90.97% after TPI, and the fault GDF fault efficiency would improve from 98.09% to 99.09%.

Due to the very large GDF ATPG CPU time consumption for circuit p66171, compared to the other circuits, the GDF ATPG CPU reduction is also less than for the ISCAS and Boolean industrial circuits. The GDF ATPG CPU times reduce from 169104 seconds without TPs to 150528 seconds after TPI; a reduction of 11.0%. Not taking into account circuit p66171, the GDF ATPG CPU times would reduce from 39469 (169104-129635) seconds without TPs to 20856 (150528-129672) seconds after TPI; a reduction of 47.2%. 47.2% is again more conform the CPU time reductions seen for the ISCAS and Boolean industrial circuits.

6.2.2 The impact of TPI for SAF ATPG on GDF test set sizes only

The results given in the previous subsection show that TPI not only results in GDF test set size reduction, but also in GDF fault coverage/efficiency improvement. The GDF fault coverage/efficiency improvement means that also new GDF patterns have to be generated for the GDFs that have become detectable after TPI. These new GDF patterns have a negative impact on the GDF test set size reduction. In order to get the GDF ATPG test set size reduction after TPI without this negative impact, Tables 6.3 and 6.4 will show experimental results in case the GDF ATPG stops generating GDF patterns when the same fault efficiency has been reached as without TPI [Geu02b]. Although, given the same fault efficiency, the GDF fault coverage can still improve after TPI (the results in Subsection 6.2.1 have shown that the GDF fault coverage improvement is much larger than the GDF fault efficiency improvement after multi-stage TPI), using this fault efficiency limit already gives a better understanding of the impact of TPI on the GDF test set size reduction itself.

In Tables 6.3 and 6.4, Column *Circuit* shows the names of the circuits. Column

Table 6.3: GDF ATPG results of ISCAS circuits with fault efficiency limitation

Circuit	GDF ATPG without TPs			GDF ATPG after TPI					
				Without FE limit			With FE limit		
	T	FC(%)	CPU	T	FC(%)	CPU	T	FC(%)	CPU
c1355	142	96.22	2.08 s	78	97.65	0.98 s	78	97.65	1.01 s
c1908	157	96.83	2.43 s	113	97.72	1.62 s	112	97.42	1.69 s
c3540	162	92.29	5.89 s	126	94.88	4.56 s	126	94.88	4.79 s
c7552	131	96.67	11.3 s	80	96.89	8.44 s	80	96.89	8.77 s
Subtotal	592	95.57	21.7 s	397	96.58	15.6 s	396	96.54	16.3 s
s1196	156	95.64	1.33 s	77	95.72	0.79 s	77	95.72	0.81 s
s1488	139	89.53	5.18 s	118	95.73	1.44 s	109	94.98	1.42 s
s1494	143	87.94	4.79 s	127	95.54	1.57 s	121	95.23	1.54 s
s5378	141	88.54	7.97 s	92	92.15	4.68 s	89	92.32	4.82 s
s9234.1	232	81.46	52.3 s	99	87.82	13.1 s	73	86.17	12.0 s
s13207.1	357	83.85	56.0 s	257	88.83	33.2 s	257	88.83	34.3 s
s15850.1	207	81.43	59.0 s	161	87.12	36.7 s	132	86.61	35.5 s
s38417	154	94.17	125 s	118	94.16	84.3 s	118	94.16	87.7 s
s38584.1	229	89.23	264 s	187	91.18	175 s	176	91.06	179 s
Subtotal	1758	88.63	576 s	1236	91.27	351 s	1152	91.04	357 s
s938a	184	78.59	1.36 s	73	77.53	0.62 s	73	77.53	0.62 s
s3330a	151	80.50	7.40 s	96	86.77	2.95 s	75	85.24	3.12 s
Subtotal	335	80.08	8.76 s	169	84.74	3.57 s	148	83.54	3.74 s
Total	2685	89.07	607 s	1802	91.61	370 s	1696	91.37	377 s

GDF ATPG without TPs shows the compact GDF ATPG results for the circuits without TPs; the GDF test set size (*T*), the GDF fault coverage (*FC(%)*), and the CPU time spent on GDF ATPG (*CPU*) are listed. The GDF fault efficiency is not listed as it will be approximately the same before TPI as after TPI.[1] Column *GDF ATPG after TPI* shows the compact GDF ATPG results for the circuits after multi-stage TPI; Column *without FE limit* shows the results as presented in the previous subsection; Column *with FE limit* shows the GDF ATPG results when the GDF ATPG stops generating GDF patterns when the same fault efficiency has been reached as for the circuits without TPs. These GDF ATPG results consist of the GDF test set size (*T*), the GDF fault coverage (*FC(%)*), and the CPU time spent on GDF ATPG (*CPU*) after TPI.

In case the GDF ATPG stops generating ATPG patterns when the same fault efficiency has been reached as for the ISCAS circuits without TPs, 106 (1802-1696) fewer patterns are generated compared to the case that there is no fault efficiency limit. Especially for

---

[1] The GDF ATPG tool is not able to stop generating patterns when exactly the same fault efficiency is reached as for the circuits without TPs. Therefore approximately the same fault efficiency.

circuits s9234.1 (-26 patterns), s15850.1 (-29 patterns), and s3330a (-21 patterns) fewer GDF patterns are generated with the fault efficiency limit.

Because fewer GDF patterns are generated due to the fault efficiency limit, the fault coverages after GDF ATPG are not as high as the GDF fault coverages without the fault efficiency limit; i.e., 91.37% with fault efficiency limit versus 91.61% without fault efficiency limit. Especially the fault coverages for circuits s1488, s9234.1, s15850.1 and s3330a with the fault efficiency limit are less than without the fault efficiency limit. This is not strange as for these circuits the number of GDF patterns has reduced the most with the fault efficiency limit. Still the GDF fault coverage is significantly higher than for the circuits without TPs, 91.37% versus 89.07%.

The CPU times do not differ much. The CPU times with the fault efficiency limit are even larger than the CPU times without fault efficiency limit, 377 seconds versus 370 seconds. This can be partially explained by ATPG CPU time variations and by the fact that it takes time to check if a certain fault efficiency has been reached.

Also for the industrial benchmark circuits fewer GDF patterns are generated in case the GDF ATPG stops generating patterns when the same fault efficiency has been reached as for the circuits without TPs; for the Boolean circuits 1149 (10,452-9303) fewer patterns are generated, and for the three-state circuits 4219 (8366-4147) fewer patterns. Especially for the three-state circuits the difference in number of GDF patterns is very large. Again the results for circuit p66171 are the cause of this large difference. Without TPs it takes the GDF ATPG tool 36 hours (129,635 seconds to be exactly) to generate GDF patterns for only 41.09% of the detectable GDF faults. After TPI, it has become much easier for the GDF ATPG tool to generate GDF patterns. In only 2005 seconds, the GDF ATPG reaches the 41% fault efficiency level after the generation of only 173 patterns. Without the fault efficiency limit, the GDF ATPG continues generating patterns until the time limit of 36 hours has been reached. At that time 2777 patterns have been generated resulting in a GDF fault coverage of 85.55%, far more than the 39.01% fault coverage after GDF ATPG with the fault efficiency limit.

Not only for circuit p66171, the number of GDF ATPG patterns has reduced significantly in case the FE limit is used. Also for circuits p162057 (-643 patterns), p104649 (-988 patterns), and p114605 (-290 patterns) the reduction is large. But as seen before, this reduction is at the cost of a lower fault coverage.

For the Boolean industrial circuits the fault coverage has been reduced from 92.54% without fault efficiency limit to 91.62% with this limit. For the three-state circuits, the GDF fault coverage has been reduced from 90.42% without fault efficiency limit to 84.98% with fault efficiency limit. This large difference for the three-state circuits is mostly caused by circuit p66171, as explained above.

The GDF ATPG CPU times with and without fault efficiency limit for the Boolean industrial circuits do not differ much; 17,448 seconds without limit compared to 17,113 seconds with the limit. On the other hand, the GDF CPU times for the three-state industrial circuits differ enormously; 167,976 seconds without fault efficiency limit compared

Table 6.4: GDF ATPG results of industrial circuits with fault efficiency limitation

Circuit	GDF ATPG without TPs			GDF ATPG after TPI					
				Without FE limit			With FE limit		
	T	FC(%)	CPU	T	FC(%)	CPU	T	FC(%)	CPU
p7653	660	90.98	107 s	465	93.24	52.9 s	435	92.73	52.8 s
p13138	134	95.35	22.8 s	110	96.10	16.1 s	110	96.10	16.8 s
p14148	623	90.50	155 s	384	89.66	116 s	382	89.63	120 s
p27811	273	82.46	301 s	151	83.47	190 s	129	83.04	194 s
p31025	1185	94.19	1343 s	1014	95.06	941 s	966	94.87	908 s
p34592	310	98.48	320 s	268	99.13	239 s	229	98.69	248 s
p36503	615	83.68	647 s	431	88.10	327 s	370	87.51	322 s
p43282	747	92.60	1078 s	580	93.63	815 s	535	93.08	836 s
p43663	296	94.24	191 s	238	94.69	157 s	235	94.68	165 s
p72767	603	93.02	5036 s	439	93.80	2019 s	342	93.07	1996 s
p73133	629	93.29	3368 s	431	94.01	2012 s	323	93.26	1921 s
p73257	5697	87.21	7389 s	4612	87.37	5725 s	4607	87.36	6153 s
p75344	477	91.92	1156 s	342	93.05	824 s	296	92.89	823 s
p162057	3578	87.47	18431 s	987	93.13	4008 s	344	90.37	3351 s
Subtotal	15827	90.50	39551 s	10452	92.54	17448 s	9303	91.62	17113 s
p32118	1189	82.50	915 s	592	85.40	440 s	463	84.30	470 s
p37021	358	38.87	185 s	240	44.27	132 s	191	43.61	133 s
p66171	1067	37.44	129635 s	2777	85.55	129672 s	173	39.01	2005 s
p71553	220	96.67	1497 s	169	96.75	1193 s	176	96.75	1147 s
p93140	668	93.88	3450 s	499	94.08	2323 s	467	93.94	2419 s
p104649	1956	89.60	11975 s	1636	94.25	6453 s	648	91.16	5519 s
p114605	2332	95.03	16641 s	1743	95.76	7758 s	1453	95.21	8239 s
p137498	1082	92.09	4802 s	710	93.22	2554 s	576	92.97	2473 s
Subtotal	8872	83.98	169104 s	8366	90.42	150528 s	4147	84.98	22408 s
Total	24699	87.37	208655 s	18818	91.52	167976 s	13450	88.42	39521 s

to 39,521 seconds with the limit. Again this is mainly caused by circuit p66171. Without the fault efficiency limit, the GDF ATPG consumes 36 hours, while with the fault efficiency limit, the GDF ATPG only consumes 2005 seconds.

Over all industrial circuits, the number of compact GDF ATPG patterns is reduced from 18818 patterns without the fault efficiency limit to 13450 patterns with the limit; in other words a reduction of 28.5%. However, this GDF pattern reduction is at the cost of a GDF fault coverage reduction. The GDF fault coverage is reduced from 91.52% without the limit to 88.42% with the limit. Still the GDF fault coverage after TPI with the fault efficiency limit is 1.05% higher than for the circuits without TPs; 88.42% versus

87.37%. The GDF ATPG CPU times in case of the fault efficiency limit are enormously reduced compared to the GDF ATPG CPU times without the fault efficiency limit. But this difference is mainly caused by circuit p66171.

6.3 Summary and conclusions

Besides facilitating compact SAF ATPG in generating compact test sets, TPI should also be able to facilitate the generation of tests for other fault models. In this chapter we have tested the impact of multi-stage TPI, as described in Chapter 5, on test sets generated by a compact GDF ATPG.

The GDF ATPG, on which the impact of TPI has been tested, is a gross GDF ATPG that uses the Transition Fault (TF) model. Experimental results of multi-stage TPI on GDF ATPG show that for both the ISCAS and the industrial benchmark circuits significant test set size reductions can be achieved; 32.9% GDF test set size reduction for the ISCAS circuits, and 34.0% for the Boolean industrial circuits. For the three-state industrial circuits 5.7% reduction has been achieved. This lower reduction is caused by the results for circuit p66171. Within the time limit for the GDF ATPG of 36 hours, 1067 patterns were generated for only 41% of the detectable faults. After TPI, the GDF ATPG was able to generate 2777 patterns for 89% of the faults within 36 hours. A lot more patterns, but also a more than twice higher GDF fault coverage. Without taking into account circuit p66171, 28.4% GDF test set size reduction is achieved for the three-state industrial circuits.

Besides a significant impact on the GDF test set sizes, multi-stage TPI also results in significant GDF fault coverage improvements. The GDF fault coverage for the ISCAS circuits improve from 89.07% to 91.61%, for the Boolean industrial circuits from 90.50% to 92.54% and for the three-state industrial circuits from 83.98% to 90.42%. The large difference in fault coverage for the three-state circuits is again caused by circuit p66171, as explained above. Without circuit p66171, the fault coverages for the three-state circuits improve from 89.20% without TPs to 90.97% after TPI.

Like for SAF ATPG, TPI also results in large GDF ATPG CPU time reduction. The GDF ATPG CPU times reduction is 39% for the ISCAS circuits, 55.9% for the Boolean industrial circuits, and 11.0% for the three-state industrial circuits. Without taking into account circuit p66171, which causes the poor CPU time reduction for the three-state circuits, the CPU reduction is 47.2%.

The GDF fault coverage improvement after TPI has a negative impact on the GDF test set size reduction. For the GDFs that have become detectable after TPI, also GDF patterns have to be generated by the ATPG. To get the impact of TPI on GDF test set sizes without the negative impact of the fault coverage improvement, also GDF ATPG runs with a limit on the fault efficiency have been performed on the circuits after TPI. The GDF ATPG has to stop generating patterns for the circuits after multi-stage TPI as soon as the same fault efficiency has been reached as for the circuits without TPs.

With the fault efficiency limit indeed better GDF test set size reductions have been achieved. Compared to GDF ATPG without the fault efficiency limit, the GDF test set sizes reduce from 1802 patterns to 1696 patterns for the ISCAS circuits, from 10,452 patterns to 9303 patterns for the Boolean industrial circuits, and from 8366 to 4147 patterns for the three-state industrial circuits. Especially the reduction for the three-state circuits is very large. Again this is for a large part caused by circuit p66171.

Due to the fault efficiency limit, of course the GDF fault coverages are not as high as without the fault efficiency limit. Still higher fault coverages have been achieved than for the circuits without TPs.

The difference in GDF CPU times between the GDF ATPG results with and without fault efficiency limit is not very high, except for the three-state industrial circuits. With the fault efficiency limit, the GDF ATPG stops generating GDF patterns for circuit p66171 after 2005 seconds, while without this limit it generates patterns for 36 hours.

The compact GDF ATPG results after TPI for SAF ATPG, i.e., multi-stage TPI, presented in this chapter, have shown that TPI is also capable of significantly reducing GDF ATPG test set sizes. Not only the GDF test set sizes reduce, also the fault coverages that can be achieved with GDF ATPG increase significantly after TPI. The increase in GDF fault coverages has a negative impact on the GDF test set size reductions. As a result, the GDF test set size reductions are not as high as the SAF ATPG test set size reductions. On the other hand, the improvements in GDF fault coverages also result in a significant improvement of the quality of the GDF tests. The GDF ATPG CPU times can be very large as shown for circuit p66171. TPI results in large GDF ATPG CPU time reductions. GDF CPU time reductions up to 78.3% for circuit p162057 have been achieved. For circuit p66171, the GDF CPU time reduction was even 99%, given the same fault efficiency before and after TPI.

Overall, the experimental results given in this chapter have shown that TPI for SAF ATPG can significantly facilitate GDF ATPG, resulting in smaller test sets, higher fault coverages and lower ATPG times.

CHAPTER 7

Summary and conclusions

This final chapter provides a summary of the main subjects of this dissertation, an overview of the main contributions, and topics for further research.

In **Chapter 1** it has been motivated that the IC-manufacturers need high quality structural tests for their integrated circuits. These high quality tests are necessary to make sure that as few as possible malfunctioning ICs are shipped to customers. These structural tests can be performed on-chip and off-chip, i.e., with testers. Off-chip testing has the advantage that high fault coverages can be achieved, but expensive testers are required with a lot of memory. For on-chip testing, or Built-In Self-Test (BIST), no expensive testers are required, but the fault coverages that can be achieved with on-chip testing are lower and on-chip testing means silicon overhead. Because the increasing complexity of nowadays ICs makes the external testers more and more costly, BIST becomes more and more interesting. But before BIST can successfully be used for a circuit, the BIST fault coverages have to be high enough to meet the test quality demands of the customers of the IC-manufacturer.

Test Point Insertion (TPI) can be used to improve the testability of circuits. By the insertion of Control Points (CPs) and Observation Points (OPs) it becomes easier to activate faults and to propagate fault effects to circuit outputs. This suggests that TPI can help to improve the fault coverage for on-chip testing, and to reduce the high costs of testers for off-chip testing, by reducing the number of test patterns and the tester memory requirements. Not every position in the circuit is suitable for Test Points (TPs). Although several TPI algorithms exist, most of them assume Boolean circuits and cannot cope with industrial circuits, with high impedance and unknown values, that are found in the real world. Inserting TPs at wrong chosen positions can result in circuit damage.

One of the targets in this dissertation is to investigate on-chip methods with which the fault coverage can be improved without a large silicon overhead, and which can cope with industrial (three-state) circuits. The other targets consist of investing existing TPI methods and developing new TPI methods for improving on-chip fault coverage and off-chip test set size reduction. These TPI methods should not only be applicable to small Boolean circuits, but also to large industrial designs. The targets described in this dissertation are summarized chapter-wise:

Chapter 2: BIST The purpose of BIST is to embed a test on-chip in order to reduce external tester requirements. However, due to Random Pattern Resistant (RPR) faults in the circuits, the fault coverages achievable with BIST are often not high enough to meet the high quality requirements of the IC industry. Also a successful BIST implementation implies several restrictions on a circuit, especially with respect to three-state circuits.

Several state-of-the-art BIST methods have been described with which higher fault coverages can be achieved. These state-of-the-art BIST methods have been divided into methods that modify the Pseudo-Random (PR) test patterns, methods that embed a full set of deterministic test patterns, and methods that only embed parts of deterministic test patterns.

The BIST methods that modify the generated test patterns or partially embed deterministic test patterns, are capable of increasing the fault coverages significantly, up to complete coverage. However, especially for RPR circuits, reaching a high or a complete fault coverage is at the cost of a significant silicon overhead, often over 10% and sometimes even over 100%. Also the BIST methods which embed full sets of deterministic patterns have significant silicon overhead, sometimes over 100%.

When the silicon overhead of BIST methods is too high for successful BIST implementation, one can consider redesigning the circuit, or the insertion of Test Points (TPs) to make the IC better testable. Although this impacts the design process, it helps improving the BIST fault coverage and reducing the silicon overhead of BIST.

Chapter 3: Test Point Insertion TPI improves the testability of a circuit. CPs can be inserted to improve the controllability of lines and OPs to improve the observability of lines. Besides the traditional AND/OR CP, the transparent scan flipflop (TSFF) is introduced as TP. A categorization of TPI methods is given; it depends on the method used to determine the faults which testability should be improved; i.e., a method based on ATPG, fault simulation or testability analysis (TA). TPI methods can also be divided into test set dependent and test set independent methods. In case of industrial circuits, TPI algorithms should be able to cope with industrial circuits and avoid bus-conflicts. Also no TPs should be allowed at the critical to avoid performance loss.

Several existing state-of-the-art TPI algorithms have been described for improving BIST fault coverage, among the Cost Reduction Factor (CRF) TPI algorithm of Seiss [Sei91], the Hybrid CRF (HCRF) TPI algorithm of Tsai [Tsa97] and the Multi-phase TPI (MTPI) algorithm of Tamarapalli [Tam96]. These three TPI algorithms assume a STUMPS based architecture, and both the CRF and HCRF TPI algorithm use a cost function, based on COP TA measures, to select the best TP positions. Therefore the STUMPS architecture, the COP TA measures, the cost function and cost gradients have been described first.

In Chapters 4 to 6 several TPI issues are addressed. An overview of these topics is given including a description of benchmark circuits, i.e., ISCAS and Philips benchmark circuits, that have been used to test the new TPI techniques and algorithms proposed in Chapters 4 to 6.

Chapter 4: Test Point Insertion for BIST In Chapters 4 to 6, the HCRF TPI algorithm has been used as a base for further TPI development. It is described why this algorithm has been chosen. The HCRF TPI algorithm assumes Boolean circuits and does not take into account any implications with respect to industrial circuits. A new TPI algorithm has been proposed, based on the HCRF TPI algorithm, that can cope with industrial circuits and results in even better BIST fault coverage improvement after TPI. Also a new cost function, the *NPAT* cost function, has been proposed for this new TPI algorithm to improve the BIST fault coverage after TPI even further. In order to cope with industrial circuits, the COP TA measures, cost function and gradients calculations have been extended such that they can be used in combination with high impedance values, unknown values, and three-state elements.

Experimental results of the proposed *NPAT* TPI algorithm for industrial circuits have shown that better BIST fault coverages can be achieved than with the original Hybrid CRF TPI algorithm of Tsai, on both Boolean and industrial (three-state) circuits. The results also show that the algorithm can be applied to small circuits and large complex industrial designs.

A method has been introduced that reduces the number of TP candidates in the circuit, such that the CPU time spent by the *NPAT* TPI algorithm is reduced without impacting the quality of the TP selection.

Chapter 5: Test point insertion for compact SAF ATPG The growth in ATPG test set sizes, caused by the increasing complexity of circuits, has a major impact on the ATE demands for the semiconductor industry. Experimental results have shown that *NPAT* TPI, that is only targeted at improving the BIST fault coverage for a circuit, is already capable of significantly reducing compact ATPG test set sizes. Also the stuck-at fault (SAF) ATPG fault coverages improve, and the ATPG CPU times reduce after *NPAT* TPI. Still several ATPG specific testability issues were not yet solved, resulting in a not optimal test set size reduction for several circuits, i.e., less than 10%. A new cost function has been proposed that takes into account test counts (TCs). TCs give a lower bound on the ATPG test set size, given complete fault coverage. High TC measures in a circuit indicate that the ATPG test set size will also be high. With the proposed TC-based cost function, the TPI algorithm will also try to reduce the TCs in the circuit in order to reduce the ATPG test set size. Experimental results show that with the TPI algorithm that takes into account TCs, i.e., TC&COP TPI, in general better test set size reduction can be achieved, i.e., 1.7% better reduction for the ISCAS benchmark circuits up to 17% better reduction for the industrial benchmark circuits.

Also after TC&COP TPI there are still circuits with disappointing test set size reductions. These circuits often suffer from very large fan-out free regions (FFRs). Although the FFR outputs have high TC values, the TC&COP TPI and *NPAT* TPI algorithm do not always insert TPs in these large FFRs to reduce the TC values. Therefore, the TPI algorithm has been extended with a TPI pre-process that inserts TPs to reduce the FFR sizes within the circuit. Four TPI methods for reducing large FFRs have been described to identify and reduce the large FFRs. With this TPI pre-process, also for the circuits that suffer from very large FFRs, the test set sizes can be reduced significantly.

Because each circuit suffers from different testability problems (related to SAF ATPG), the multi-stage TPI algorithm has been developed. The TPI process is divided into multiple stages. In each stage a TPI pre-process is applied, that analyzes the TA measures to select a cost function aimed at solving the testability problems of that specific circuit. With this cost function several TPs are inserted, after which the TPI pre-process is run again to select a new cost function. After multi-stage TPI, that also includes TPI for reducing large FFRs, extra reductions of 4.5% for the ISCAS circuits and 12.2% for the industrial circuits have been achieved compared to TC&COP TPI.

Chapter 6: Test Point Insertion for delay fault ATPG Besides facilitating SAF ATPG in generating compact test sets, TPI should also be able to facilitate the generation of tests for other fault models. The impact of TPI for SAF ATPG, i.e., multi-stage TPI, has been tested on gate-delay fault (GDF) ATPG, i.e., Transition Fault ATPG. Experimental results have shown that indeed significant GDF test set size reductions can be achieved with TPI for SAF ATPG; on average 32.9% for the ISCAS circuits and 23.8% for the industrial circuits. For several circuits also the GDF fault coverage increased considerably. This increase in GDF ATPG fault coverage has a negative impact on the GDF test set size reduction; the GDF ATPG also has to generate patterns for the GDFs that have become detectable after TPI. Due to the much larger GDF fault coverage increase than SAF fault coverage increase, the GDF test set size reduction was not as high as the SAF test set size reduction.

Altering the experimental setup such that the GDF ATPG stops generating patterns for the circuits with TPs as soon as the same fault efficiency has been reached as for the circuits without TPs, results in a reduction of the negative impact of the GDF fault coverage improvement. The experimental results have shown that indeed better GDF test set size reductions have been achieved. The GDF test set size reduction increased to 36.8% for the ISCAS circuits and to 45.5% for the industrial circuits.

Besides reducing the GDF test set sizes and improving the GDF fault coverage, TPI also resulted in significant GDF ATPG CPU time reductions, up to 99%.

7.1 Major contributions

The major contributions of this research are:

On-chip testing (BIST) [Chapter 2]

1. Investigation and categorization of existing state-of-the art on-chip test methods, i.e., BIST methods.
2. An analysis and comparison of the existing BIST methods with respect to hardware overhead and pseudo-random fault coverage. In general will random pattern resistant circuits result in (too) large silicon overhead for BIST methods to be applicable.
3. An analysis of the existing BIST methods whether they are only applicable to small benchmark circuits, or also to large industrial designs.

Test Point Insertion for BIST

1. A categorization of TPI methods [Section 3.3].
2. Investigation of existing state-of-the-art TPI algorithms [Section 3.7].
3. An analysis and comparison of three TPI algorithms that improve pseudo random fault coverage with respect to [Section 4.1]:
    - the number of inserted test points versus the pseudo random fault coverage,
    - the application on Boolean and industrial circuits,
    - the application on large circuits.
4. The extension of the Hybrid CRF TPI algorithm such that it can be used for improving the pseudo random fault coverage of both Boolean as well as industrial (three-state circuits). This has been accomplished by adding three-state and unknown value capabilities to [Sections 4.2,4.3][Geu03]:
    - COP,
    - cost function,
    - cost gradients.
5. The development of a new cost function for the extended Hybrid CRF TPI method for industrial circuits in order to further improve the pseudo-random fault coverage after TPI [Section 4.3][Geu03].
6. The development of a method to reduce the number of test point candidates that results in a TPI CPU time reduction without impacting the quality of the test point selection [Section 4.3][Geu02a].

Test Point Insertion for stuck-at fault ATPG

1. An evaluation of the impact of TPI for BIST on reducing stuck-at fault ATPG test set sizes [Section 5.1][Geu00].
2. An analysis of which testability problems cause large ATPG test sets, i.e., large FFRs and high test counts [Sections 5.2,5.4][Geu00, Geu02b].
3. The development of a new TPI algorithm that takes into account test counts such that the TPI algorithm becomes more aimed at reducing ATPG test set sizes. This is accomplished by [Section 5.3][Geu00, Geu01]:
    - defining a heuristic for propagating test counts in a circuit with fan-outs,
    - the development of a cost function that takes into account both COP and test counts testabilities,
    - defining the test counts gradient equations,
    - extending the HCRF TPI algorithm for industrial circuits with an extra event-driven forward and backward propagation step of changed test counts due to a TP.
4. The development of a TPI pre-process to reduce large FFRs in a circuit as large FFRs often cause large stuck-at fault ATPG test sets [Section 5.4][Geu02b].
5. The development of a multi-stage TPI algorithm. In each stage of the TPI process, a cost function is selected, based on testability analysis measures, that targets the highest testability problems found in the circuit, given already inserted test points in previous stages [Section 5.5][Geu02b].

TPI for gate-delay fault ATPG

1. The application of TPI has been tested on another fault model, the gross gate-delay fault. An analysis is given of the impact of multi-stage TPI, on gate-delay fault ATPG, i.e., transition fault ATPG. The following gate-delay fault ATPG results have been analyzed before and after multi-stage TPI [Section 6.2][Geu02b]:
    - The gate-delay fault test set size reduction, together with the gate-delay fault coverage improvement, and the gate-delay fault ATPG CPU time reduction.
    - The gate-delay fault test set size reduction only, as the gate-delay fault coverage improvement has a negative impact on the gate-delay fault test set size reduction.

Experimental results have demonstrated that the proposed TPI for BIST algorithm, i.e., *NPAT* TPI, is applicable to small and large designs, both Boolean and industrial, and that in general better fault coverage improvements can be achieved that with existing state-of-the-art TPI algorithms [Geu03]. The pseudo random fault coverage for industrial circuits has been improved from 93.53% with the original Hybrid CRF TPI algorithm for industrial circuits to 94.49% with *NPAT* TPI, see Table 4.5. In other words 15% of the faults that were not covered after TPI with the original cost function, have become covered after *NPAT* TPI.

NPAT TPI also facilitates stuck-at fault ATPG in reducing ATPG test sets, improving the ATPG fault coverage and reducing the ATPG CPU times [Geu00]. On average, the ATPG test set sizes of the Boolean and industrial benchmark circuits have been reduced with 43%, respectively 26%. The presented TPI for SAF ATPG algorithms, i.e., TC&COP TPI and multi-stage TPI (including the TPI for reducing large FFRs preprocess) significantly improve the facilitation of SAF ATPG compared to *NPAT* TPI [Geu00, Geu01, Geu02b]. On average, the stuck-at fault ATPG test set sizes have been reduced with 47% for the Boolean, respectively 51% for the industrial circuits compared to the circuits without test points.

It has been shown that multi-stage TPI does not only facilitates stuck-at fault ATPG, but also gate-delay fault ATPG [Geu02b]. The average gate-delay fault ATPG test set size reduction after TPI was 37% for the Boolean, respectively 46% for the industrial circuits, given the same gate-delay fault efficiency before and after TPI. Also the impact of TPI on the gate-delay fault coverage and ATPG times is significant. For circuit p66171, the gate-delay fault ATPG time for generating patterns for 41% of all detectable faults, is reduced from 36 hours without TPI to 2005 seconds after TPI. After TPI, the gate-delay fault ATPG was able to generate in 36 hours patterns for 89% of all detectable faults, compared to the 39% for the circuit without test points.

The proposed new TPI algorithms and techniques, i.e., *NPAT* TPI, TC&COP TPI, TPI for reducing large FFRs, multi-stage TPI, and the TPI candidates reduction technique, have been implemented and integrated in the Delft Advanced Test (DAT) generation system and AMSAL, and are being used worldwide within Philips as part of their logic test tool set. The robustness and application of the TPI algorithms on very large industrial designs have been proven at Philips as the TPI algorithms are currently applied to multi-million gate designs.

7.2 Suggested topics for future research

A state-of-the art TPI tool has been developed. It improves PR fault coverage for on-chip testing, and significantly reduces compact SAF and GDF ATPG test set sizes for both Boolean and three-state circuits. However, for both on- and off-chip testing, there are still several TPI related topics for which we suggest further research.

With respect to TPI for BIST, i.e., on-chip-testing, we suggest the following topics:

1. The integration of the Hybrid TPI algorithm for industrial circuits with a state-of-the-art BIST method and comparing the BIST fault and silicon overhead with other state-of-the-art TPI+BIST solutions, i.e., MTPI BIST.
2. The development of TPI extensions for industrial circuits in order to:
   - insert TPs for bypassing fixed circuit inputs, i.e., inputs from embedded memories.
   - add CPs between buses and circuit outputs, such that it can be prevented that unknown values, caused by bus-conflicts, can corrupt the MISR state.

With respect to TPI for ATPG, we suggest the following topics:

1. Research and integration of other TA measures that indicate on large ATPG test set sizes, such that the ATPG test set sizes can be further reduced. The integration of ATPG data taken from an ATPG run for the circuit without TPs to find the ATPG ’problem’ locations in the circuit, hence the best TP positions in the circuit.
2. An evaluation of the impact of TPI on other fault models than SAF and GDF ATPG, e.g., path-delay faults or bridge-faults.
3. Research of a GDF related TA measure and a TPI cost function that is specifically aimed at reducing GDF ATPG test set sizes.

With respect to TPI in general, i.e., both for on-chip and off-chip testing, we suggest the following topics:

1. Further reducing the number of TPI candidates without impacting the test quality, i.e., test set size reduction and/or fault coverage improvement.
2. Research how TPI can be optimally used for partial-scan circuits, i.e., circuits that contain SFFs but also non-scannable memory elements. In other words, TPI for controlling and observing non-scannable FFs, without impacting the critical path, hence the performance of the IC.
3. Research how TPI can help making untestable/redundant faults testable in test mode, without impacting the purpose of the redundancy, when such a purpose exists.
4. An investigation of the impact on the test set size reduction and/or fault coverage improvement when TPs on critical paths are avoided.

APPENDIX A

ISCAS ’85 and ’89 benchmark circuits

Tables A.1 and A.2 provide characteristics of the ISCAS’85 [Brg85], ISCAS’89 [Brg89] and ISCAS’89 addendum [Glo93] circuits that have been used in this dissertations for benchmarking. The ISCAS’85 circuits consist of only combinational logic circuits, while the sequential ISCAS’89 circuits are used in full-scan mode. All circuits are Boolean designs and contain no three-state elements.

Column *Circuits* gives the name of the circuit. Columns *#Conn* and *#Gates* show the number of connections respectively the number of gates. Columns *#GPI* and *#GPO* show the number of PIs+SFF outputs, respectively the number of POs+SFF inputs. Column FFR_{max} shows the number of inputs of the largest FFR found in the circuit.

Tables A.3 and A.4 provide compact stuck-at fault (SAF) and gate-delay fault (GDF) ATPG data for the ISCAS’85 respectively the ISCAS’89 (addendum) circuits listed in the Tables A.1 and A.2. This information consists of the number of test patterns in the test set (*T*), the fault coverage (*FC(%)*), fault efficiency (*FE(%)*) and the CPU time spent on compact SAF respectively GDF ATPG (*CPU*). The ATPG runs took place on a 1600+

Table A.1: Characteristics of several ISCAS’85 benchmark circuits

Circuit	#Conn	#Gates	#GPI	#GPO	FFR_{max}
c880	880	383	60	26	17
c1355	1355	546	41	32	16
c1908	1908	880	33	25	51
c2670	2670	1193	233	140	72
c3540	3540	1669	50	22	62
c5315	5315	2307	178	123	19
c6288	6288	2416	32	32	4
c7552	7552	3512	207	108	35

Table A.2: Characteristics of several ISCAS’89 (addendum) benchmark circuits

Circuit	#Conn	#Gates	#GPI	#GPO	FFR_{max}
s1196	1196	529	32	32	29
s1423	1423	657	91	79	20
s1488	1488	653	14	25	75
s1494	1494	647	14	25	76
s5378	5295	2779	214	228	122
s9234.1	9234	5597	247	250	165
s13207.1	13179	7951	700	790	103
s15850.1	15847	9772	611	684	75
s35932	35612	16065	1763	2048	11
s38417	38339	22179	1664	1742	59
s38584.1	38432	19253	1464	1730	90
s499a	477	152	23	44	5
s635a	635	286	34	33	32
s938a	938	446	66	33	134
s1269a	1261	569	55	47	18
s1512a	1494	780	86	78	32
s3271a	3252	1572	142	130	42
s3330a	3330	1789	172	205	176
s3384a	3375	1685	226	209	94
s4863a	4799	2342	153	120	14
s6669a	6518	3080	322	294	12

AMD Athlon XP computer, with 512MB DDR SDRAM memory running Redhat Linux 7.3.

Tables A.5 and A.6 provide PR fault simulation run data for the ISCAS’85 respectively the ISCAS’89 (addendum) circuits. The first columns (*Complete run*) show the results after the complete fault simulation run, consisting of the number of applied PR patterns (T), the fault coverage (*FE(%)*), fault efficiency (*FE(%)*) and the CPU time spent on applying T PR patterns. The last three columns (*FE* $\geq$ *99%*) show the number of applied PR pattern after which the fault efficiency is higher than 99% (T) and the fault coverage (*FE(%)*) and fault efficiency (*FE(%)*) corresponding to this number of applied PR patterns.

Table A.3: ATPG characteristics of the ISCAS’85 circuits

Circuit	Stuck-at fault ATPG				Gate-delay fault ATPG			
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)	CPU
c880	28	100.00	100.00	0.27 s	34	97.35	97.35	0.37 s
c1355	89	99.70	100.00	0.49 s	142	96.22	96.53	2.08 s
c1908	114	99.71	100.00	1.32 s	157	96.83	97.07	2.43 s
c2670	59	96.40	100.00	1.30 s	61	91.85	94.89	2.12 s
c3540	119	96.38	100.00	2.59 s	162	92.29	95.74	5.89 s
c5315	78	99.42	100.00	2.47 s	85	97.29	97.99	4.49 s
c6288	32	99.46	100.00	2.77 s	77	99.22	99.90	14.1 s
c7552	128	98.55	100.00	6.01 s	131	96.67	97.83	11.3 s

Table A.4: ATPG characteristics of the ISCAS’89 (addendum) circuits

Circuit	Stuck-at fault ATPG				Gate-delay fault ATPG			
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)	CPU
s1196	133	100.00	100.00	0.52 s	156	95.64	95.73	1.33 s
s1423	39	99.09	100.00	0.35 s	72	86.31	97.17	1.15 s
s1488	115	100.00	100.00	0.53 s	139	89.53	97.08	5.18 s
s1494	117	99.46	100.00	0.52 s	143	87.94	96.01	4.79 s
s5378	117	98.87	100.00	2.81 s	141	88.54	96.97	7.97 s
s9234.1	135	93.95	99.91	20.0 s	232	81.46	92.86	52.3 s
s13207.1	272	98.87	100.00	15.5 s	357	83.85	96.39	56.0 s
s15850.1	128	97.51	100.00	13.9 s	207	81.43	96.30	59.0 s
s35932	14	89.69	100.00	20.8 s	24	83.03	97.05	139 s
s38417	88	99.68	100.00	30.9 s	154	94.17	96.12	125 s
s38584.1	123	95.57	100.00	44.9 s	229	89.23	96.90	264 s
s499a	104	100.00	100.00	0.28 s	66	68.43	99.78	0.34 s
s635a	47	100.00	100.00	0.23 s	99	73.47	100.00	0.53 s
s938a	145	100.00	100.00	0.45 s	184	78.59	96.09	1.36 s
s1269a	38	100.00	100.00	0.33 s	54	94.38	95.22	0.71 s
s1512a	64	100.00	100.00	0.41 s	74	84.12	90.51	2.67 s
s3271a	59	100.00	100.00	1.03 s	77	94.70	97.51	3.08 s
s3330a	163	100.00	100.00	2.20 s	151	80.50	90.90	7.40 s
s3384a	74	100.00	100.00	1.04 s	96	98.89	98.91	2.89 s
s4863a	48	100.00	100.00	1.94 s	87	96.77	97.78	9.10 s
s6669a	35	100.00	100.00	2.29 s	59	99.68	99.68	5.76 s

Table A.5: PR Fault simulation results of the ISCAS’85 circuits

Circuit	Complete run				FE $\geq$ 99%		
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)
c880	9024	100.00	100.00	0.18 s	2720	99.58	99.58
c1355	2848	99.49	100.00	0.16 s	1312	98.60	99.11
c1908	11520	99.52	100.00	0.47 s	2080	98.67	99.15
c2670	32000	84.82	89.08	2.05 s	-	-	-
c3540	32000	95.97	99.97	2.30 s	1408	95.01	99.01
c5315	3712	98.90	100.00	0.63 s	416	98.00	99.10
c6288	128	99.56	100.00	0.49 s	64	99.33	99.77
c7552	32000	95.30	97.03	6.17 s	-	-	-

Table A.6: PR Fault simulation results of the ISCAS’89 (addendum) circuits

Circuit	Complete run				FE $\geq$ 99%		
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)
s1196	32000	99.60	99.60	0.43 s	12864	99.03	99.03
s1423	17568	99.08	100.00	0.35 s	2176	98.22	99.14
s1488	3520	100.00	100.00	0.18 s	1536	99.13	99.13
s1494	3520	99.20	100.00	0.17 s	1312	98.21	99.00
s5378	32000	98.94	99.80	3.88 s	6816	98.13	99.00
s9234.1	32000	87.05	93.49	10.0 s	-	-	-
s13207.1	32000	97.05	98.58	13.0 s	-	-	-
s15850.1	32000	92.83	96.14	15.4 s	-	-	-
s35932	224	89.81	100.00	1.05 s	96	89.06	99.26
s38417	32000	94.42	94.95	46.5 s	-	-	-
s38584.1	32000	95.31	99.46	43.2 s	14080	94.85	99.00
s499a	32000	41.85	41.85	0.36 s	-	-	-
s635a	32000	73.57	73.57	0.29 s	-	-	-
s938a	32000	64.98	64.98	0.48 s	-	-	-
s1269a	1056	100.00	100.00	0.14 s	256	99.33	99.33
s1512a	32000	95.28	95.28	0.64 s	-	-	-
s3271a	32000	99.82	99.82	2.08 s	5984	99.24	99.24
s3330a	32000	86.97	86.97	2.93 s	-	-	-
s3384a	32000	96.18	96.18	2.23 s	-	-	-
s4863a	32000	97.57	97.57	3.29 s	-	-	-
s6669a	4416	100.00	100.00	0.85 s	96	99.22	99.22

APPENDIX B

Industrial benchmark circuits

Tables B.1 and B.2 provide characteristics of the Philips industrial cores used in this dissertation for benchmarking. Most of these industrial circuits are used in products, except for circuit p93140, which only has been designed for test purposes. The circuits listed in Table B.1 are Boolean circuits, while the circuits listed in Table B.2 contain one or more three-state elements.

Column *Circuits* gives the name of the circuit. Columns *#Conn* and *#Gates* show the number of connections respectively the number of gates. Columns *#GPI* and *#GPO* show

Table B.1: Characteristics of several Boolean Philips industrial benchmark circuits

Circuit	#Conn	#Gates	#GPI	#GPO	FFR_{max}
p5973	6621	2979	245	225	35
p7653	7653	3685	197	264	67
p13138	13138	5901	498	440	52
p14148	14148	6827	559	572	170
p27811	27811	13247	1273	1403	96
p31025	31025	14913	629	662	137
p34592	34592	17685	408	651	92
p36503	36503	19292	1502	1514	96
p43282	43282	21484	1003	906	110
p43663	43663	17865	8232	8249	56
p72767	72767	35266	1923	1630	169
p73133	73133	35675	1922	1629	169
p73257	73257	37587	2353	2229	269
p75344	75344	36561	3716	3742	181
p162057	162057	77994	5914	5841	1518

Table B.2: Characteristics of several tri-state Philips industrial benchmark circuits

Circuit	#Conn	#Gates	#GPI	#GPO	FFR$_{max}$
p32118	32118	14449	1572	1560	244
p37021	37021	17059	2001	1843	96
p66171	66192	27011	1002	1047	78
p71553	71553	32691	762	520	19
p93140	93140	47809	3331	2891	474
p104649	104649	44619	3533	3466	244
p114605	114605	57356	2754	2670	131
p137498	137498	59240	6984	6592	433
p481470	481470	237328	18564	18577	1044
p596922	596922	313711	17768	17356	467
p598004	598004	285355	20648	20084	1518
p705050	705704	359615	30856	29916	256
p824184	824184	453002	26124	24548	520
p854266	855659	433361	32194	32303	292

the number of PIs+SFF outputs, respectively the number of POs+SFF inputs. Column FFR_{max} shows the number of inputs of the largest FFR found in the circuit.

Tables B.3 and B.4 provide compact SAF and GDF ATPG data for the Boolean respectively three-state industrial circuits listed in the previous tables. This information consists of the number of test patterns (T), the fault coverage (FC), fault efficiency (FE) and the CPU time spent on compact SAF, respectively GDF ATPG, (CPU). Each run had a CPU time limit of 36 hours. Results with runs which took more than 36 hours are incomplete, as ATPG did not finish yet. Because of the long GDF ATPG times and the limited memory capacity (512Mb) of the test computer, GDF ATPG has only been performed on circuits with less than 200,000 signal lines. The ATPG runs took place on a 1600+ AMD Athlon XP computer, with 512MB DDR SDRAM memory running Redhat Linux 7.3.

Tables B.5 and B.6 provide PR fault simulation run data for the Boolean respectively three-state industrial circuits listed in the previous tables. Note that the PR results are only shown for the three-state circuits, which do not contain buses that can become in conflict due to the application of PR patterns!

The first columns (*Complete run*) show the results after the complete fault simulation run, consisting of the number of applied PR patterns (T), the fault coverage (*FC(%)*), fault efficiency (*FE(%)*) and the CPU time spent after T PR patterns. The last three columns ($FE \geq 99\%$) show the number of applied PR pattern after which the fault efficiency is higher than 99% (T) and the fault coverage (*FC(%)*) and fault efficiency (*FE(%)*) corresponding to this number of applied PR patterns.

Table B.3: ATPG characteristics of the Boolean industrial circuits

Circuit	Stuck-at fault ATPG				Gate-delay fault ATPG			
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)	CPU
p5973	58	98.93	99.83	5.17 s	90	92.80	95.11	76.4 s
p7653	334	99.48	99.99	14.9 s	660	90.98	95.04	107 s
p13138	66	99.76	100.00	5.21 s	134	95.35	95.88	22.8 s
p14148	385	99.99	100.00	30.0 s	623	90.50	94.76	155 s
p27811	200	95.21	99.97	42.8 s	273	82.46	96.64	301 s
p31025	714	97.90	99.64	307 s	1185	94.19	96.37	1343 s
p34592	243	99.88	99.90	142 s	310	98.48	98.50	320 s
p36503	175	97.20	100.00	71.3 s	615	83.68	91.31	647 s
p43282	382	97.66	99.83	207 s	747	92.60	96.39	1078 s
p43663	194	99.23	100.00	58.8 s	296	94.24	99.27	191 s
p72767	438	96.31	99.09	1527 s	603	93.02	96.91	5036 s
p73133	407	96.37	99.08	1492 s	629	93.29	97.11	3368 s
p73257	2097	98.06	99.97	1030 s	5697	87.21	89.08	7389 s
p75344	273	99.18	99.90	153 s	477	91.92	96.99	1156 s
p162057	2055	98.51	99.59	926 s	3578	87.47	93.06	18431 s

Table B.4: ATPG characteristics of the three-state industrial circuits

Circuit	Stuck-at fault ATPG				Gate-delay fault ATPG			
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)	CPU
p32118	570	93.50	99.95	131 s	1189	82.50	97.01	915 s
p37021	531	76.89	99.99	56.1 s	358	38.87	98.43	185 s
p66171	2042	96.79	100.00	576 s	1067	37.44	41.09	129635 s
p71553	138	96.69	99.98	295 s	220	96.67	99.91	1497 s
p93140	511	93.81	99.18	1139 s	668	93.88	99.17	3450 s
p104649	1031	98.96	99.70	847 s	1956	89.60	94.77	11975 s
p114605	689	98.44	99.55	827 s	2332	95.03	98.52	16641 s
p137498	446	98.66	99.93	308 s	1082	92.09	98.73	4802 s
p481470	1679	99.23	99.93	16560 s	-	-	-	-
p596922	2038	96.87	99.74	8887 s	-	-	-	-
p598004	2129	98.70	99.76	5051 s	-	-	-	-
p705050	655	76.76	99.87	8463 s	-	-	-	-
p824184	2066	96.54	99.82	130153 s	-	-	-	-
p854266	565	97.54	99.75	6881 s	-	-	-	-

Table B.5: PR Fault simulation results of the Boolean industrial circuits

Circuit	Complete run				FE $\geq$ 99%		
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)
p5973	7328	98.93	100.00	1.65 s	3520	97.94	99.01
p7653	32000	98.48	99.14	6.66 s	29888	98.36	99.02
p13138	736	99.58	100.00	0.62 s	192	98.58	99.01
p14148	32000	94.27	94.29	18.2 s	-	-	-
p27811	32000	91.90	96.29	31.5 s	-	-	-
p31025	32000	95.15	97.09	64.1 s	-	-	-
p34592	32000	94.79	94.82	36.7 s	-	-	-
p36503	32000	89.03	94.89	47.4 s	-	-	-
p43282	32000	93.27	95.29	51.6 s	-	-	-
p43663	32000	98.64	99.63	60.0 s	896	98.01	99.01
p72767	32000	91.73	95.19	257 s	-	-	-
p73133	32000	92.13	95.52	194 s	-	-	-
p73257	32000	59.14	61.53	156 s	-	-	-
p75344	32000	98.12	99.03	90.8 s	29056	98.11	99.01
p162057	32000	96.11	97.17	310 s	-	-	-

Table B.6: PR Fault simulation results of the three-state industrial circuits

Circuit	Complete run				FE $\geq$ 99%		
	T	FC(%)	FE(%)	CPU	T	FC(%)	FE(%)
p32118	32000	84.07	90.61	41.4 s	-	-	-
p37021	32000	67.95	91.78	45.2 s	-	-	-
p114605	32000	89.22	90.47	202 s	-	-	-
p137498	32000	93.79	95.49	205 s	-	-	-
p481470	32000	82.82	83.73	999 s	-	-	-
p596922	32000	86.16	89.57	1102 s	-	-	-

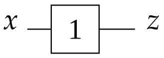
APPENDIX C

Boolean elements

The gate symbol, truth table and the COP TA measures are given for several Boolean primitive elements. The symbols and truth tables are taken from [vdL96]. The truth tables do take into account the ’high impedance’ value Z and the ’unknown’ value U such that they are compatible with three-state circuits. In case of (N)AND and (N)OR gate, the truth tables are shown for a two-input gate.

For all Boolean elements, the COP TA measures, extended for three-state logic, consist of the equations for $C0$, $C1$ and W. All CZs, WZ^0, and WZ^1 are 0, because Z values cannot propagate through Boolean elements; they result in U values on the outputs. CU can be found with $CU = 1 - C1 - C0$.

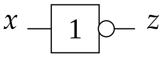
Buffer:



z	x			
	0	1	Z	U
	0	1	U	U

$C0_z$	$C0_x$
$C1_z$	$C1_x$
W_x	W_z

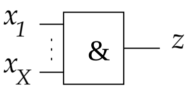
Inverter:



z	x			
	0	1	Z	U
	1	0	U	U

$C0_z$	$C1_x$
$C1_z$	$C0_x$
W_x	W_z

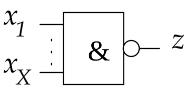
AND:



z		x_1			
		0	1	Z	U
x_2	0	0	0	0	0
	1	0	1	U	U
	Z	0	U	U	U
	U	0	U	U	U

$C0_z$	$1 - \prod_{i=1}^{X}(1 - C0_{x_i})$
$C1_z$	$\prod_{i=1}^{X}(C1_{x_i})$
W_{x_j}	$W_z \cdot \prod_{i=1,i\neq j}^{X}(C1_{x_i})$

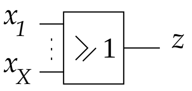
NAND:



z		x_1			
		0	1	Z	U
x_2	0	1	1	1	1
	0	1	0	U	U
	Z	1	U	U	U
	U	1	U	U	U

$C0_z$	$\prod_{i=1}^{X}(C1_{x_i})$
$C1_z$	$1-\prod_{i=1}^{X}(1-C0_{x_i})$
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(C1_{x_i})$

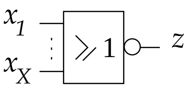
OR:



z		x_1			
		0	1	Z	U
x_2	0	0	1	U	U
	1	1	1	1	1
	Z	U	1	U	U
	U	U	1	U	U

$C0_z$	$\prod_{i=1}^{X}C0_{x_i}$
$C1_z$	$1-\prod_{i=1}^{X}(1-C1_{x_i})$
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(C0_{x_i})$

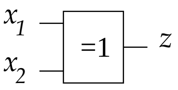
NOR:



z		x_1			
		0	1	Z	U
x_2	0	1	0	U	U
	1	0	0	0	0
	Z	U	0	U	U
	U	U	0	U	U

$C0_z$	$1-\prod_{i=1}^{X}(1-C1_{x_i})$
$C1_z$	$\prod_{i=1}^{X}C0_{x_i}$
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(C0_{x_i})$

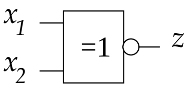
XOR:



z		x_1			
		0	1	Z	U
x_2	0	0	1	U	U
	1	1	0	U	U
	Z	U	U	U	U
	U	U	U	U	U

$C0_z$	$C0_{x_1}\cdot C0_{x_2}+C1_{x_1}\cdot C1_{x_2}$
$C1_z$	$C0_{x_1}\cdot C1_{x_2}+C1_{x_1}\cdot C0_{x_2}$
W_{x_1}	$W_z\cdot(C0_{x_2}+C1_{x_2})$

NXOR:



z		x_1			
		0	1	Z	U
x_2	0	1	0	U	U
	1	0	1	U	U
	Z	U	U	U	U
	U	U	U	U	U

$C0_z$	$C0_{x_1}\cdot C1_{x_2}+C1_{x_1}\cdot C0_{x_2}$
$C1_z$	$C0_{x_1}\cdot C0_{x_2}+C1_{x_1}\cdot C1_{x_2}$
W_{x_1}	$W_z\cdot(C0_{x_2}+C1_{x_2})$

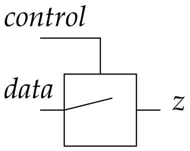
APPENDIX D

Three-state elements

The gate symbol, truth table and the COP testability analysis measures are given for several three-state primitive elements. The truth tables are taken from [vdL96]. The truth tables for the buses assume a two-input bus.

For all listed three-state elements, the COP TA measures consist of the equations for $C0$, $C1$, CZ, W, WZ^0 and WZ^1. CU can be found with $CU = 1 - C1 - C0 - CZ$.

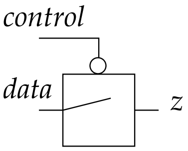
Switch:



z		control			
		0	1	Z	U
d	0	Z	0	U	U
a	1	Z	1	U	U
t	Z	Z	Z	Z	Z
a	U	Z	U	U	U

$C0_z$	$C1_{control} \cdot C0_{data}$
$C1_z$	$C1_{control} \cdot C1_{data}$
CZ_z	$C0_{control} + CZ_{data} - C1_{control} \cdot CZ_{data}$
W_{data}	$W_z \cdot C1_{control}$
WZ^0_{data}	$WZ^0_z \cdot C1_{control}$
WZ^1_{data}	$WZ^1_z \cdot C1_{control}$
$W_{control}$	$WZ^0_z \cdot C0_{data} + WZ^1_z \cdot C1_{data}$
$WZ^0_{control}$	0
$WZ^1_{control}$	0

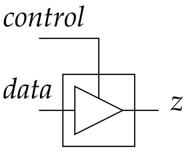
NSwitch:



z		control			
		0	1	Z	U
d	0	0	Z	U	U
a	1	1	Z	U	U
t	Z	Z	Z	Z	Z
a	U	U	Z	U	U

$C0_z$	$C0_{control} \cdot C0_{data}$
$C1_z$	$C0_{control} \cdot C1_{data}$
CZ_z	$C1_{control} + CZ_{data} - C1_{control} \cdot CZ_{data}$
W_{data}	$W_z \cdot C0_{control}$
WZ^0_{data}	$WZ^0_z \cdot C0_{control}$
WZ^1_{data}	$WZ^1_z \cdot C0_{control}$
$W_{control}$	$WZ^0_z \cdot C0_{data} + WZ^1_z \cdot C1_{data}$
$WZ^0_{control}$	0
$WZ^1_{control}$	0

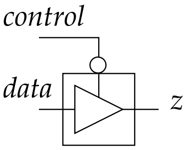
Bus-driver:



z		control			
		0	1	Z	U
d	0	Z	0	U	U
a	1	Z	1	U	U
t	Z	Z	U	U	U
a	U	Z	U	U	U

$C0_z$	$C1_{control} \cdot C0_{data}$
$C1_z$	$C1_{control} \cdot C1_{data}$
CZ_z	$C0_{control}$
W_{data}	$W_z \cdot C1_{control}$
WZ^0_{data}	0
WZ^1_{data}	0
$W_{control}$	$WZ^0_z \cdot C0_{data} + WZ^1_z \cdot C1_{data}$
$WZ^0_{control}$	0
$WZ^1_{control}$	0

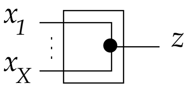
NBus-driver:



z		control			
		0	1	Z	U
d	0	0	Z	U	U
a	1	1	Z	U	U
t	Z	U	Z	U	U
a	U	U	Z	U	U

$C0_z$	$C0_{control} \cdot C0_{data}$
$C1_z$	$C0_{control} \cdot C1_{data}$
CZ_z	$C1_{control}$
W_{data}	$W_z \cdot C0_{control}$
WZ^0_{data}	0
WZ^1_{data}	0
$W_{control}$	$WZ^0_z \cdot C0_{data} + WZ^1_z \cdot C1_{data}$
$WZ^0_{control}$	0
$WZ^1_{control}$	0

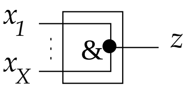
Three-state bus:



z		x_1			
		0	1	Z	U
x_2	0	0	U	0	U
	1	U	1	1	U
	Z	0	1	Z	U
	U	U	U	U	U

$C0_z$	$\prod_{i=1}^{X}(C0_{x_i}+CZ_{x_i})-\prod_{i=1}^{X}(CZ_{x_i})$
$C1_z$	$\prod_{i=1}^{X}(C1_{x_i}+CZ_{x_i})-\prod_{i=1}^{X}(CZ_{x_i})$
CZ_z	$\prod_{i=1}^{X}(CZ_{x_i})$
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$
$WZ_{x_j}^0$	$WZ_z^0\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$
$WZ_{x_j}^1$	$WZ_z^1\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$

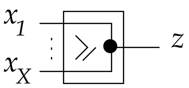
Wired AND:



z		x_1			
		0	1	Z	U
x_2	0	0	0	0	0
	1	0	1	1	U
	Z	0	1	Z	U
	U	0	U	U	U

$C0_z$	$1-\prod_{i=1}^{X}(1-C0_{x_i})$
$C1_z$	$\prod_{i=1}^{X}(C1_{x_i}+CZ_{x_i})-\prod_{i=1}^{X}(CZ_{x_i})$
CZ_z	$\prod_{i=1}^{X}(CZ_{x_i})$
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(C1_{x_i}+CZ_{x_i})$
$WZ_{x_j}^0$	$W_z\cdot\prod_{i=1,i\neq j}^{X}(C1_{x_i}+CZ_{x_i})$
	$+(WZ_z^0-W_z)\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$
$WZ_{x_j}^1$	$WZ_z^1\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$

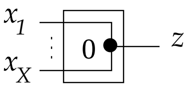
Wired OR:



z		x_1			
		0	1	Z	U
x_2	0	0	1	0	U
	1	1	1	1	1
	Z	0	1	Z	U
	U	U	1	U	U

$C0_z$	$\prod_{i=1}^{X}(C0_{x_i}+CZ_{x_i})-\prod_{i=1}^{X}(CZ_{x_i})$
$C1_z$	$1-\prod_{i=1}^{X}(1-C1_{x_i})$
CZ_z	$\prod_{i=1}^{X}(CZ_{x_i})$
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(C0_{x_i}+CZ_{x_i})$
$WZ_{x_j}^0$	$WZ_z^0\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$
$WZ_{x_j}^1$	$W_z\cdot\prod_{i=1,i\neq j}^{X}(C0_{x_i}+CZ_{x_i})$
	$+(WZ_z^1-W_z)\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$

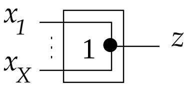
Pull-down bus:



z		x_1			
		0	1	Z	U
x_2	0	0	U	0	U
	1	U	1	1	U
	Z	0	1	0	U
	U	U	U	U	U

$C0_z$	$\prod_{i=1}^{X}(C0_{x_i}+CZ_{x_i})$
$C1_z$	$\prod_{i=1}^{X}(C1_{x_i}+CZ_{x_i})-\prod_{i=1}^{X}(CZ_{x_i})$
CZ_z	0
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$
$WZ_{x_j}^0$	0
$WZ_{x_j}^1$	$W_z\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$

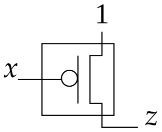
Pull-up bus:



z		x_1			
		0	1	Z	U
x_2	0	0	U	0	U
	1	U	1	1	U
	Z	0	1	1	U
	U	U	U	U	U

$C0_z$	$\prod_{i=1}^{X}(C0_{x_i}+CZ_{x_i})-\prod_{i=1}^{X}(CZ_{x_i})$
$C1_z$	$\prod_{i=1}^{X}(C1_{x_i}+CZ_{x_i})$
CZ_z	0
W_{x_j}	$W_z\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$
$WZ_{x_j}^0$	$W_z\cdot\prod_{i=1,i\neq j}^{X}(CZ_{x_i})$
$WZ_{x_j}^1$	0

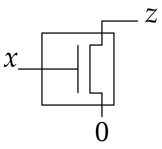
Open-drain PFET:



z	x			
	0	1	Z	U
	1	Z	U	U

$C0_z$	0
$C1_z$	$C0_x$
CZ_z	$C1_x$
W_x	WZ_z^1
WZ_x^0	0
WZ_x^1	0

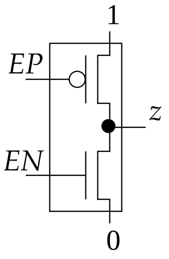
Open-drain PFET:



z	x			
	0	1	Z	U
	Z	0	U	U

$C0_z$	$C1_x$
$C1_z$	0
CZ_z	$C0_x$
W_x	WZ_z^0
WZ_x^0	0
WZ_x^1	0

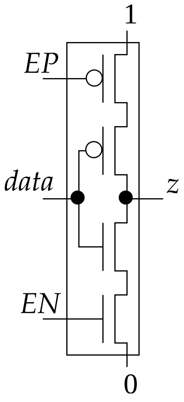
Tri:



z		EP			
		0	1	Z	U
	0	1	Z	U	U
E	1	U	0	U	U
N	Z	U	U	U	U
	U	U	U	U	U

$C0_z$	$C1_{EP} \cdot C1_{EN}$
$C1_z$	$C0_{EP} \cdot C0_{EN}$
CZ_z	$C1_{EP} \cdot C0_{EN}$
W_{EN}	$WZ_z^0 \cdot C1_{EP}$
WZ_{EN}^0	0
WZ_{EN}^1	0
W_{EP}	$WZ_z^1 \cdot C0_{EN}$
WZ_{EP}^0	0
WZ_{EP}^1	0

CMOS Tristate inverter (Trinv):



$C0_z$	$C1_{data} \cdot C1_{EN}$
$C1_z$	$C0_{data} \cdot C0_{EP}$
CZ_z	$C0_{data} \cdot C1_{EP} + C1_{data} \cdot C0_{EN} + C1_{EP} \cdot C0_{EN}$ $-(C0_{data} + C1_{data}) \cdot C1_{EP} \cdot C0_{EN}$
W_{data}	$WZ_z^1 \cdot C0_{EP} \cdot C0_{EN} + WZ_z^0 \cdot C1_{EP} \cdot C1_{EN}$ $+W_z \cdot C0_{EP} \cdot C1_{EN}$
WZ_{data}^0	0
WZ_{data}^1	0
W_{EP}	$WZ_z^1 \cdot C0_{data}$
WZ_{EP}^0	0
WZ_{EP}^1	0
W_{EN}	$WZ_z^0 \cdot C1_{data}$
WZ_{EN}^0	0
WZ_{EN}^1	0

$z = bus[nswitch[data, ODP[EP]], switch[data, ODN[EN]]]$

APPENDIX E

Delft Advanced Test (DAT) generation system and AMSAL

The TPI algorithms and techniques in this dissertation have been implemented in C++ inside the ATPG system: *DAT*, the Delft Advanced Test generation system. Details on the implementation can be found in [Kon96a]. The code has been successfully ported to HP (PA-Risc; HP-UX) and Linux platforms.

The DAT tool can be used fully stand-alone, and can be operated by means of (scripts or command line input in) a dedicated command language (DCL [vdL96] [Kon96b]). It can read in circuits in the ISCAS formats, 2 internal formats, KISS2 [Kis89], BLIF [Bli89], Verilog (limited), and the (Philips) NDL format (and via AMSAL, also EDIF 1.1.0 and EDIF 2.0.0).

The main tasks which DAT can perform are ATPG (using (pseudo-)random and deterministic (extended FAN-algorithm based) test pattern generation (TPG)), fault simulation (FS), logic simulation (LS) and test point insertion (TPI). The circuits may contain one or more scan-chains, which can be explicitly or implicitly present in the circuit. By means of the command language, very complex ATPG and TPI schemes (having multiple stages, each using multiple strategies in test generation) can be defined, and 'aliased' into a simple user-defined command. Apart from Boolean gates, various 3-state elements such as (regular 3-state, 'wired', and 'pulled') buses, and various types of switches and drivers, as well as bidirectional terminals (I/O pins) are supported. Random TPG, FS and LS can be performed by 'parallel patterns' (32 patterns at the same time). Compact ATPG by means of extended additional targets TPG is supported as well. Fault model/test methods combinations supported for combinational circuits are: Single Stuck-At Fault (SSAF) logic observation testing, and potential logic observation testing, and I_{DDq} testing of SSAFs and bridging faults. TPI for improving PR fault coverage can be performed by using the CRF TPI algorithm [Sei91] for Boolean circuits and TPI for both improving PR fault coverage and reducing ATPG test time and data volume for both Boolean and three-state circuits can be performed by enhanced TPI algorithms based on the Hybrid CRF algorithm [Tsa97].

Since early 1995, the combinational circuit part of DAT has replaced the heart of Philips' AMSAL [Hap93] ATPG system. The various interfacing procedures of AMSAL have been recoded in such a way that the external interface to AMSAL remains practically the same as it used to be, such as net-list compilation, pattern conversions, etc. Since 2001, the TPI part of DAT has also been embedded in AMSAL. AMSAL is used as the ATPG tool within Philips design and production departments worldwide. AMSAL in its turn, is an important part of the larger test generation system, that also incorporates test generation algorithms for various types of RAM, ROM, etc.

Glossary

This glossary gives an overview of the abbreviations and symbols used throughout this dissertation.

Abbreviations

TA	Testability analysis
ACRF	Actual cost reduction factor
ATE	Automated test equipment
ATPG	Automatic test pattern generator
BIST	Built-in self-test
CA	Cellular automaton
ChPol	Characteristic polynomial
COP	Controllability observability program
CP	Control point
CRF	Cost reduction factor
CUT	Circuit-under-test
DL	Defect level
DPM	Defects per million
F-O	Force-observe
FC	Fault coverage
FE	Fault efficiency
FF	Flipflop
FFR	Fan-out free region
FRU	Field replaceable unit
FSM	Finite-state-machine
FT	Final time frame

GDF	Gate-delay fault
HCRF	Hybrid cost reduction factor
IC	Integrated circuit
IT	Initial time frame
LFSR	Linear feedback shift register
LSSD	Level sensitive scan design
MISR	Multiple input shift register
MTPI	Multi-phase test point insertion
OP	Observation point
ORA	Output response analyzer
PFS	Probabilistic fault simulation
PGC Logic	Pattern generation and control logic
PI	Primary input
PO	Primary output
PR	Pseudo random
PRTPG	Pseudo random test pattern generator
PXTPG	Pseudo exhaustive test pattern generator
ROM	Read-only memory
RPR	Random pattern resistant
SA0	Stuck-at 0
SA1	Stuck-at 1
SAF	Stuck-at fault
SCOAP	Sandia controllability/observability analysis program
SFF	Scan flipflop
SFN	Scan, fix and normal
SGL	Sequence generating logic
SML	Sequence modifying logic
SRSG	Shift register sequence generator
STUMPS	Self-test using MISR/parallel SRSG
TC	Test count
TF	Transition fault
TP	Test point

TPG	Test pattern generator
TPI	Test point insertion
TSFF	Transparent scan flipflop
XTPG	Exhaustive test pattern generator

Symbols

$\#CP$	Number of control points
$\#TP$	Number of test points
$\#WP$	Number of observation points
Φ	Number of phases
ϕ	Phase
C	COP controllability
$C0$	COP 0-controllability
$C1$	COP 1-controllability
CM	Cube mapping
CP_{max}	Maximum number of control points
CU	COP U-controllability
Cu	Test cube
CZ	COP Z-controllability
dK/dC	Derivative of the cost with respect to a change in controllability
$dK/dC0$	Derivative of the cost with respect to a change in 0-controllability
$dK/dC1$	Derivative of the cost with respect to a change in 1-controllability
dK/dCZ	Derivative of the cost with respect to a change in Z-controllability
$dK/dE0$	Derivative of the cost with respect to a change in essential zero counts
$dK/dT0$	Derivative of the cost with respect to a change in total zero counts
$dK/dT1$	Derivative of the cost with respect to a change in essential one counts
$dK/dT1$	Derivative of the cost with respect to a change in total one counts
dK/dW	Derivative of the cost with respect to a change in observability
dK/dWZ^0	Derivative of the cost with respect to a change in Z↔0 observability
dK/dWZ^1	Derivative of the cost with respect to a change in Z↔1 observability
$E0$	Essential zero count
$E1$	Essential one count

F	Number of faults
f	Fault
h_r	Feedback coefficient for memory cell r
iCu	Image cube
IM	Identity matrix
K	Cost function
K_m	Cost of modified circuit, i.e., circuit with test points
L	Number of signal lines
l	Line
M_r	Memory cell r
$MBPC$	Minimum-benefit-per-cost
OP_{max}	Maximum number of observation points
p	Probability
Pd_f	COP detection probability of fault f
R	Number of memory cells in a shift register
r	Index of memory cells
$SC0$	SCOAP 0-controllability
$SC1$	SCOAP 1-controllability
sCu	Source cube
SW	SCOAP observability
T	Number of test patterns
t	Test pattern
$T0$	Total zero count
$T1$	Total one count
T_c	Characteristic matrix
T_{min}	Minimum test set size
TP_{max}	Maximum number of test points
W	COP observability
w	Test pattern width
WZ^0	COP Z↔0 observability
WZ^1	COP Z↔1 observability
$X(t)$	State/Pattern at time t

Bibliography

[Ake87] S.B. Akers, C. Joseph and B.Krishnamurthy. On the role of independent fault sets in the generation of minimal test sets. *Proc. of the IEEE Int. Test Conf.*, pages 1100–1107, 1987.

[Als94] M.F. Alshaibi and C.R. Kime. Fixed-biased pseudo-random built-in selft-test for random pattern resistant circuits. *Proc. of the IEEE Int. Test Conf.*, pages 929–938, 1994.

[Bar87] P.H Bardell, W.H. McAnney and J. Savir. *Built-In Pseudo-Random Testing of Digital Circuits*, chapter 8. John Wiley & Sons, New York, 1987.

[Bli89] Berkeley logic interchange format. http://www-cad.eecs.berkeley.edu/Respep/Research/sis/abstract.html, 1989.

[Bra84] R.K. Brayton, G.D. Hachtel C. McMullen, A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, Boston, 1984.

[Bra89] R.K. Brayton, Rudell R., A. Sangiovanni-Vincentelli, A.R.Wang. An exact minimizer for boolean relations. *Proc. of IEEE Int. Conf. on Computer Aided Design (ICCAD)*, pages 316–319, 1989.

[Brg84] F. Brglez. On testability analysis of combinational networks. *Proc. of Int. Symp. on Circuits and Systems*, pages 221–225, 1984.

[Brg85] F. Brglez and H. Fujiwara. A neural netlist of 10 combinational benchmark designs and a special translator in fortran. *Proc. of Int. Symp. on Circuits and Systems*, June 1985.

[Brg89] F. Brglez et al. Combinational profiles of sequential benchmark circuits. *Proc. of Int. Symp. on Circuits and Systems*, pages 1929–1934, May 1989.

[Cha92] Jau-Shien Chang and Chen-Shang Lin. Test set compaction for combinational circuits. *Proc. of Asian Test Symp.*, pages 20–25, November 1992.

[Che95a] C. Chen, and S. Gupta. A methodology to design efficient bist test pattern generators. *Proc. of the IEEE Int. Test Conf.*, pages 814–823, 1995.

[Che95b] K-T. Cheng and C-J. Lin. Timing-driven test point insertion for full-scan and partial-scan bist. *Proc. of the IEEE Int. Test Conf.*, pages 506–514, 1995.

[Chr75] N. Christofedes and K. Korman. A computational survey of methods for the set covering problem. *Management Science*, 21(5):591–599, 1975.

[Duf91] C. Dufaza and G. Cambon. Lfsr based deterministic and pseudo-random test pattern generator structures. *Proc. of European Test Conf.*, pages 27–34, 1991.

[Duf93] C. Dufaza and C. Chevalier. Lfsrom basic principle and bist application. *Proc. of European Conf. on Design Automation*, pages 211–216, 1993.

[Eic77] E.B. Eichelberger, and T.W. Williams. A logic design structure for lsi testability. *Proc. of the 14th Design Automation Conf.*, pages 462–468, June 1977.

[Fei99] C. Feige, M.J. Geuzebroek, J. ten Pierick and H. Vranken. Lbist evaluation report. Technical report, Philips Semiconductors B.V., December 1999.

[Fei01] C. Feige, and M.J. Geuzebroek. Logic bist technology evaluation: an industrial case study. *European Test Workshop - Informal Digest*, pages 333–340, 2001.

[Fuj85] H. Fujiwara. *Logic Testing and Design for Testability*. The MIT Press, Cambridge, Massachusetts, 1985.

[Geu97a] M.J. Geuzebroek. The deterministic bist scheme. Master's thesis, Delft University of Technology, Faculty of Electrical Engineering, Mekelweg 4, 2628 CD, Delft, April 1997.

[Geu97b] M.J. Geuzebroek. Test point insertion techniques. Technical Report 1-68340-44(1997)13, Delft University of Technology, Faculty of Electrical Engineering, Mekelweg 4, 2628 CD, Delft, November 1997.

[Geu00] M.J. Geuzebroek, J.Th. van der Linden and A.J. van de Goor. Test point insertion for compact test sets. *Proc. of the IEEE Int. Test Conf.*, pages 292–301, 2000.

[Geu01] M.J. Geuzebroek, A.J. van de Goor and J.Th. van der Linden. Facilitating automatic test pattern generators using test point insertion. *World Market Series Business Briefing: Global Semiconductor Manufacturing Technology*, pages 149–152, Januari 2001.

[Geu02a] M.J. Geuzebroek. Tpi facilitation techniques. Technical report, Delft University of Technology, Faculty of Electrical Engineering, Mekelweg 4, 2628 CD, Delft, November 2002.

[Geu02b] M.J. Geuzebroek, J.Th. van der Linden and A.J. van de Goor. Test point insertion that facilitates atpg in reducing test time and data volume. *Proc. of the IEEE Int. Test Conf.*, pages 138–147, 2002.

[Geu03] M.J. Geuzebroek, A.J. van de Goor and J.Th. van der Linden. Tpi for improving pr fault coverage of boolean and three-state circuits. *To appear at the European Test Workshop 2003*, pages xx–xx(6 pages), 2003.

[Glo93] C. Gloster. Dynamic scan testing: Investigating a new paradigm. Technical report, MCNC, Center of Mictroelectronics, 1993. INTERNET: http://www.cbl.ncsu.edu/pub/Benchmark_dirs/ISCAS89/ADDENDUM93/DOCUMENTATION/main.ps.

[Goe81] P. Goel, and B.C. Rosales. Podem-x: An automatic test generation system for vlsi logic circuits. *Proc. of 18th Design Automation Conf.*, pages 260–268, 1981.

[Gol80] L.H. Goldstein and E.L. Thigpen. Scoap: Sandia controllability/observability analysis program. *Proc. of 17th Design Automation Conf.*, pages 190–196, 1980.

[Gu 01] X. Gu , S.S. Chung, F. Tsang, J.A. Tofte and H. Rahmanian. An efforminimized logic bist implementation method. *Proc. of the IEEE Int. Test Conf.*, pages 1002–1010, 2001.

[Hap93] F. Hapke and R. Reche. Amsal reference manual. Technical Report Release 01.08.00, Philips GmbH, VALCO RHW, Stresemannallee 101, D-2000 Hamburg 54, Germany, September 1993.

[Har93] J. Hartmann, and G. Kemnitz. How to do weighted random testing for bist. *Proc. of IEEE Int. Conf. on Computer Aided Design (ICCAD)*, pages 568–571, 1993.

[Hay74] J.P. Hayes and A.D. Friedman. Test point placement to simplify fault detection. *IEEE Transactions on Computers*, C-33:727–735, July 1974.

[Hel92] S. Hellebrand, S. Tarnick, J. Rajski and B. Courtois. Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers. *Proc. of the IEEE Int. Test Conf.*, pages 120–129, 1992.

[Het99] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan and J. Rajski. Logic bist for large industrial designs: Real issues and case studies. *Proc. of the IEEE Int. Test Conf.*, pages 358–367, 1999.

[Hor89] P.D. Hortensius, R.D. McLeod, W. Pries, D.M. Miller and H.C. Card. Cellular automata-based pseudorandom number generators for built-in-self-test. *IEEE Transactions on Computer-Aided Design*, 8(8):842–859, 1989.

[Kag96] D. Kagaris, S. Tragoudas and A. Majumdar. Deterministic test pattern reproduction by a counter. *Proc. of European Design and Test Conf.*, pages 37–41, 1996.

[Kie98] G. Kiefer and H-J. Wunderlich. Deterministic bist with multiple scan chains. *Proc. of the IEEE Int. Test Conf.*, pages 1057–1064, 1998.

[Kie00] G. Kiefer, H. Vranken, E.J. Marinissen and H-J. Wunderlich. Application of deterministic logic bist on industrial circuits. *Proc. of the IEEE Int. Test Conf.*, pages 105–114, 2000.

[Kis89] Logic synthesis and optimization benchmarks. http://www.cbl.ncsu.edu/pub/Benchmark_dirs/LGSynth89/DOCUMENTATION, 1989.

[Kon96a] M.H. Konijnenburg and J.Th. van der Linden. Data structures and algorthims of the delft automatic test pattern generation system (dat). Technical Report 1-68340-44(1996)06, Delft University of Technology (Faculty of Electrical Engineering, Department CARDIT), Mekelweg 4, 2628 CD Delft, March 1996.

[Kon96b] M.H. Konijnenburg and J.Th. van der Linden. *Delft Automatic Test Pattern Generation Platform (DAT) User Manual*. Delft University of Technology (Faculty of Electrical Engineering, Department CARDIT), Mekelweg 4, 2628 CD Delft, March 1996.

[Kon96c] M.H. Konijnenburg, J.Th. van der Linden and A.J. van de Goor. Accelerated compact test set generation for three-state circuits. *Proc. of the IEEE Int. Test Conf.*, pages 29–38, October 1996.

[Kon98] M.H. Konijnenburg. *Automatic Test Pattern Generation for Synchronous Sequential Circuits*. PhD thesis, Delft University of Technology, Faculty of Electrical Engineering, Mekelweg 4, 2628 CD, Delft, December 1998.

[Kri87] B. Krishnamurthy. A dynamic programming approach to the test point insertion problem. *Proc. of 24th ACM/IEEE Design Automation Conf.*, pages 695–705, 1987.

[Lem94] M. Lempel, S.K. Gupta and M. Breuer. Test embedding with discrete logarithms. *Proc. of 12th IEEE VLSI Test Symposium*, pages 74–80, 1994.

[Lis87] R. Lisanke, F. Brglez, A.J. Degeus and D. Gregory. Testability-driven random test-pattern generation. *IEEE Transactions on Computer-Aided Design*, CAD-6:1082–1087, 1987.

[Maj94] A. Majumdar. A new procedure for weighted random built-in self-test. *Proc. of IEEE Int. Conf. on Computer Aided Design (ICCAD)*, pages 288–291, 1994.

[Mur90] F. Muradali, V.K. Argarwal and B. Nadeau-Dostie. New procedure for weighted random built-in-self-test. *Proc. of the IEEE Int. Test Conf.*, pages 660–669, 1990.

[Nag95] P. Nagvajara and P. Kumhon. Built-in self-test based on pre-stored test. Private Communication, Drexel University Philadelphia, 1995.

[Nak99] M. Nakao, S. Kobayashi, K. Hatayama, K. Iijima and S. Terada. Low overhead test point insertion for scan-based bist. *Proc. of the IEEE Int. Test Conf.*, pages 348–357, 1999.

[Pat91] S. Pateras and J. Rajski. Generation of correlated random patterns for the complete testing of synthesized multi-level circuits. *Proc. of 28th ACM/IEEE Design Autom. Conf.*, pages 347–352, 1991.

[Poh78] S.C. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance. *IEEE Trans. Information Theory*, IT-24(1):106–110, 1978.

[Pom91] I. Pomeranz, L.N. Reddy and S.M. Reddy. Compactest: A method to generate compact test sets for combinational circuits. *Proc. of the IEEE Int. Test Conf.*, pages 194–203, 1991.

[Pom93] I. Pomeranz and S.M. Reddy. 3-weight pseudo-random test generation based on a deterministic test set for combinational and sequential circuits. *IEEE Transactions on Computer-Aided Design*, 12(7):1050–1058, 1993.

[Pra95] G. Pramadono. Dat's path delay fault atpg. Master's thesis, Delft University of Technology, Faculty of Electrical Engineering, Mekelweg 4, 2628 CD, Delft, August 1995.

[Raj99] J. Rajski, G. Mrgugalski and J. Tyszer. Comparison study of ca-based prpgs and lfsrs with phase shifters. *Proc. of 17th IEEE VLSI Test Symposium*, pages 236–245, 1999.

[Ree96] B. Reeb and H-J. Wunderlich. Deterministic pattern generation for weighted random pattern testing. *Proc. of European Test Conf.*, pages 30–36, March 1996.

[Sav91] Y. Savaria, M. Youssef, B. Kaminska and M. Koudil. Automatic test point insertion for pseudo-random testing. *Proc. of International Symposium on Circuits and Systems*, pages 1960–1963, 1991.

[Sch95] C. Schotten and H. Meyr. Test point insertion for an area efficient bist. *Proc. of the IEEE Int. Test Conf.*, pages 515–523, 1995.

[Sei91] B.H. Seiss, P.M. Trouborst and M.H. Schulz. Test point insertion for scan-based bist. *Proc. of 2nd European Test Conf.*, pages 253–262, 1991.

[Sem01] Semiconductor Industry Association. International technology roadmap for semi-conductors 2001 edition, test and test equipment. http://public.itrs.net/Files/2001ITRS/Test.pdf, 2001.

[Sen92] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton and A. Sangiovanni-Vincentelli. Sequential circuit design using synthesis and optimization. *Proc. of Int. Conf. on Computer Design*, pages 328–333, 1992.

[Smi85] G.L. Smith. Model for delay faults based upon paths. *Proc. of the IEEE Int. Test Conf.*, pages 342–349, 1985.

[Tam96] N. Tamarapalli and J. Rajski. Constructive multi-phase test point insertion for scan-based bist. *Proc. of the IEEE Int. Test Conf.*, pages 649–658, 1996.

[Tou95] N.A. Touba and E.J. McCluskey. Transformed pseudo-random patterns for bist. *Proc. of 13th IEEE VLSI Test Symposium*, pages 410–416, 1995.

[Tou96] N.A. Touba and E.J. McCluskey. Test point insertion based on path tracing. *Proc. of 14th IEEE VLSI Test Symposium*, pages 2–8, 1996.

[Tro91] Gert-Jan Tromp. Minimal test sets for combinational circuits. *Proc. of the IEEE Int. Test Conf.*, pages 204–209, 1991.

[Tsa97] H.-C. Tsai, K.-T. Cheng, C.-J. Lin and S. Bhawmik. A hybrid algorithm for test point selection for scan-based bist. *Proc. of 34th Design Automation Conf.*, pages 478–483, 1997.

[Vas93] B. Vasudevan, D.E. Ross, M. Gala and K.L. Watson. Lfsr based deterministic hardware for at-speed bist. *Proc. of 11th IEEE VLSI Test Symposium*, pages 201–207, 1993.

[vdG98] A.J. van de Goor. *Testing Semiconductor Memories, Theory and Practice*. ComTex Publishing, http://cardit.et.tudelft.nl/˜vdgoor, 1998.

[vdL96] J. Th. van der Linden. *Automatic Test Pattern Generation for Three-State Circuits*. PhD thesis, Delft University of Technology, Faculty of Electrical Engineering, Mekelweg 4, 2628 CD, Delft, May 1996.

[Vra02] H. Vranken, F. Meister and H-J. Wunderlich. Combining deterministic logic bist with test point insertion. *Proc. of the Seventh IEEE European Test Workshop*, pages 105–110, 2002.

[Wai89] J.A. Waicukauski, E. Lindblook, E.B. Eichelberger and O.P. Forlenza. A method for generating weighted random test patterns. *IBM J. Res. Develop.*, 33(2):149–161, March 1989.

[Wai90] J.A. Waicukauski, P.A.Shupe, D.J. Giramma and A. Matin. Atpg for ultra-large structured designs. *Proc. of the IEEE Int. Test Conf.*, pages 44–51, 1990.

[Wil81] T.W. Williams, and N.C. Brown. Defect level as function of fault coverage. *IEEE Transactions on Computers,* C-030:987–988, 1981.

[Wun96] H.-J. Wunderlich and G. Kiefer. Bit-flipping bist. *Proc. of IEEE Int. Conf. on Computer Aided Design (ICCAD)*, pages 337–343, 1996.

[You93] M. Youssef, Y. Savaria and B. Kaminska. Methodology for efficiently inserting and condensing test points. *IEEE Proceedings-E*, 140(3):154–160, 1993.

Samenvatting (Summary in Dutch)

Test punt toevoeging voor het verbeteren van BIST prestaties, en het reduceren van ATPG test tijd en data volume

Dit proefschrift behandelt het vergemakkelen van structureel testen van geïntegreerde circuits (chips) op mogelijke defecten. Tijdens het fabricage-proces van chips, kunnen defecten optreden, zoals extra of ontbrekende verbindingen door stofdeeltjes, waardoor de chips niet meer volgens verwachting werken. Om te voorkomen dat chips met defecten toch in consumenten producten terechtkomen, worden structurele testen uitgevoerd door testers. Echter, door de steeds toenemende complexiteit van chips, worden de kosten van deze testers steeds hoger. Moore's law dicteert dat de snelheid (klokfrequentie) en de geheugencapaciteit van de testers steeds groter moet worden. Deze steeds grotere tester kosten gaan een steeds belangrijker aandeel vormen in de prijzen van chips.

Een alternatief voor het uitvoeren van de tests door externe testers is het insluiten van de test op de chip zelf, *Built-In Self-Test (BIST)* genaamd. Hierdoor is er geen snelle tester met heel veel geheugen meer vereist, maar kan de chip ook door een veel goedkopere (tragere en met minder geheugen toegeruste) tester worden gestart. De chip hoeft alleen maar aangestuurd worden om zichzelf te testen en na de test behoeft alleen maar 'gevraagd' te worden of de chip defecten bezit.

In Hoofdstuk 2 worden verscheidene bestaande BIST methodes beschreven en gecategoriseerd, waarbij wordt gekeken naar de foutdekking, extra silicium oppervlak en de bruikbaarheid op grote industriële chips.

Een belangrijk nadeel van BIST is dat bij moeilijk testbare circuits, of de foutdekking vaak een stuk lager is dan bij het extern testen, of de extra benodigde chip oppervlak te groot is. Een manier om de testbaarheid van chips te verbeteren is *Test Punt Toevoeging (Test Point Insertion (TPI))*, waardoor de foutdekking bij BIST verbetert en ook de benodigde extra chip oppervlak afneemt. Hoofdstuk 3 beschrijft de algemene werking van TPI en verdeelt de methodes die gebruikt worden voor het bepalen van de posities van de *test punten (TPs)*, in categorieën. Niet elk circuit, of elke positie op de chip, is geschikt voor een TP. In de industrie komen naast Boolean elementen ook vaak circuits met 'three-state' elementen voor. Deze geven nog meer restricties. In Hoofdstuk 3 wordt ook een beschrijving gegeven van bestaande TPI algoritmes. Ook wordt een overzicht gegeven van de TPI onderwerpen in de hoofdstukken 4 tot 6 en de circuits die zijn gebruikt voor het testen van de TPI algoritmes.

Hoofdstuk 4 begint met het selecteren van een bestaand TPI algoritme voor verdere ontwikkeling. Een nadeel van dit algoritme is dat het is ontworpen voor Boolean circuits en niet voor industriële circuits. Dit algoritme wordt aangepast voor complexe industriële circuits. Daarnaast wordt het TPI algoritme ook aangepast voor een nog grotere foutdekking na TPI, en wordt een methode gepresenteerd waarmee de processor tijd gebruikt door TPI, significant wordt gereduceerd, zonder dat de kwaliteit van de gekozen TP posities negatief wordt beïnvloed.

De tests voor op de testers worden gegenereerd door *Automatische Test Patroon Generators (ATPGs)*. TPI zorgt niet alleen voor een hogere foutdekking bij BIST, doch ook voor kleinere ATPG test sets, omdat het makkelijker wordt om voor de te dekken fouten test patronen te genereren. Hierdoor wordt het ook makkelijker om meerdere fouten te laten dekken door een test patroon en daardoor het totaal aantal testpatronen te verkleinen. In Hoofdstuk 5 wordt aangetoond dat het BIST TPI algoritme voor industriële circuits, gepresenteerd in Hoofdstuk 4, al goed in staat is de test sets flink te verkleinen, waarbij tevens de foutdekking toeneemt en de ATPG processor tijd nodig om de tests te genereren fors afneemt. Echter niet voor alle circuits wordt een even goede reductie van het aantal test patronen gehaald. Er zijn specifieke ATPG gerelateerde test problemen die het TPI algoritme niet weet op te lossen. In Hoofdstuk 5 wordt beschreven hoe *test counts (TCs)*, die een schatting geven voor de minimale grootte van de test, in het TPI algoritme kunnen worden geïntegreerd om de grootte van de ATPG test sets nog meer te verkleinen. Een ander probleem voor de ATPG zijn zeer grote fan-out vrije gebieden in de circuits. Deze zorgen vaak voor grote test sets en moeten daardoor gedurende TPI verkleind worden. Hiervoor worden vier methoden gepresenteerd die gebruikt kunnen worden voor het verkleinen van deze fan-out vrije gebieden. Omdat elk circuit zijn eigen specifieke test problemen heeft, wordt een nieuwe TPI algoritme gepresenteerd dat bestaat uit meerdere stadia. In elke stadium wordt eerst uitgezocht welk test probleem of problemen het belangrijkst zijn om te worden aangepakt. Het TPI algoritme zal dan enkele TPs toevoegen om dit test probleem aan te pakken. In het volgende stadium wordt opnieuw bekeken welk test probleem het belangrijkst is om aan te pakken. Resultaten in Hoofdstuk 5 laten zien dat hiermee het aantal test patronen nog verder gereduceerd kan worden.

In Hoofdstuk 6 wordt aangetoond dat TPI niet alleen werkt voor het vergemakkelen en verkleinen van ATPG test sets voor het *stuck-at fault (SAF)* foutmodel, maar ook voor het *gate-delay fault (GDF)* model. De GDF test sets zijn vaak nog groter dan de SAF test sets, waardoor TPI ook voor dit foutmodel een belangrijke rol kan spelen. Uit de resultaten blijkt dat niet alleen het aantal GDF test patronen significant afneemt, maar ook dat de foutdekking behoorlijk toeneemt na TPI. Ook de processor tijd voor GDF ATPG neemt fors af.

Tot slot wordt in Hoofdstuk 7 een samenvatting gegeven van het onderzoek, worden de belangrijkste bijdrages opgesomd en worden aanbevelingen gedaan voor verder onderzoek.

Jeroen Geuzebroek

Curriculum Vitae

Jeroen Geuzebroek was born in Rotterdam on June 18, 1974. After his secondary education at the R.S.G. (currently "Maerlant College") in Brielle from 1986 to 1992, he studied at the department of Electrical Engineering of the Delft University of Technology, where he graduated in 1997. During his masters study, he investigated the advantages and disadvantages of state-of-the-art BIST implementations, i.e., BIST implementations that embed deterministic test patterns. One deterministic BIST method, the Deterministic LFSR was implemented in the DAT test tool, developed at the Delft University of Technology. In 1997 he started this PhD study at the computer architecture group of Prof. A.J. van de Goor. This research was funded by Philips Semiconductors. The software developed during this research has been integrated in Philips' production test system worldwide. During his PhD period, Jeroen also joined Philips for several months for the evaluation of a BIST tool. Jeroen's (research) interests include Design-for-Test, all aspects of facilitating testing, especially with respect to test data reduction and compression, and the development of software for solving complex problems.