Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Test Compression Based on the Illinois-Scan Architecture

Daniel Kralík

Department of Computer Science
Supervisor: doc. Ing. Petr Fišer, Ph.D.

August 3, 2020

Acknowledgements

I would like to thank my supervisor Petr Fišer, who was my supervisor. His lectures and comments were very helpful to my bachelor thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on August 3, 2020 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Kralík, Daniel. *Test Compression Based on the Illinois-Scan Architecture.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Bakalářská práce se zabývá návrhem a implementací algoritmu pro kopresi testu založeném na Illinois-Scan dekompresní architektuře používané pro testování digitálních obvodů. Jedná se o metodu testování sekvenčích obvodů, která za účelem snadné testovatelnosti převede sekvenční obvody do spojené kombinační logiky a klopných obvodů (DFFs), které jsou následně přerozděleny do vybraných scan-chainů. Testovací vektory jsou pak vedené paralelně několika scan-chainy, což může způsobit neúplné pokrytí všech poruch. Proto je pro nás užitečné hledat takové configurace scan-chainů, které maximalizují pokrytí poruch a minimalizují délku testu. K tomuto účelu máme k dispozici nástroj na generování testu (ATPG) - nakonec jsem použil ATPG Atalanta. Výsledky práce jsou zhodnoceny v závěru.

Klíčová slova komprese testu, Illinois-Scan, scan-chain, pokrytí poruch, digitální obvody, kombinační obvody , sekvenční obvody, klopné obvody, DFF, ATPG

Abstract

The Bachelor thesis aims to design and implement a test compression algorithm based on the Illinois-Scan decompression architecture used for digital circuits. This is a method for a sequential logic testing, which converts sequential circuits in the connected combinational logic and flip-flops (DFFs), which are reordered in the chosen scan-chains. The test vectors are delivered to multiple scan-chains in parallel, which may cause the fault coverage loss. Therefore is very useful to find the configuration of the scan-chains, which maximizes the faults coverage and minimizes the test length. For this purpose, we use available test generation tools (ATPGs) - finally, I used ATPG Atalanta. The outcomes of the research are evaluated in the conclusion.

Keywords test compression, Illinois-Scan, scan-chain, faults coverage, digital circuits, combinational circuits, sequential circuits, flip-flops, DFF, ATPG

Contents

List of Figures

Introduction

Digital circuits have come a part of our everyday life since the 20th century. We probably could not see them at first sight but every technology from the electric toothbrush to the supercomputer in NASA is based on digital circuits. Chips in our tools are created from a semiconductor wafer, which has contained thousands and thousands of logic gates and latches. Although this part of information technology is used to be neglected between programmers and developers, just decades of development of the digital circuits has brought technologies into the form, how can we see today. So, we have got many reasons to study them.

In the beginning, I would like to explain, why is useful to be aimed at testing digital circuits. Chips are created from semiconductor wafers. Let the 300mm (circle diameter) wafer gives 1cm2 chip, then material costs (wafer, copper, etc...) gives 5%, fab (fabrication facility) operational costs gives 25%, personal costs gives 20% but testing costs gives 40% [1]. According to other source, testing costs gives 30%[2]. So, testing is a big share of chip's costs. In my bachelor thesis, I will be focused on one type of sequential logic testing algorithm based on Illinois-Scan Architecture.

The Bachelor thesis aims to design and implement a test compression algorithm, which tests sequential circuits. In my work, I am focused on the method called Illinois-Scan Architecture. Digital circuits are tested with extensive test data volume. The algorithm is saving test data volume (test compression) and decreases test application time[1]. On the other hand, our approach may cause fault coverage loss.

The input of our program is sequential or combinational logic (BLIF). According to our configuration file (generated or handmade) with scan-chains, our program generates a new file based on Illinois-Scan Architecture (BLIF). The testing of this file with ATPG is evaluated in the experimental part. In the experimental part we find out, how could distribution of the scan-chains affects fault coverage and testing time and volume. The outcomes of experimental measurement are summarized in the conclusion.

Structure

Bachelor thesis is divided into the six chapters with conclusion.

- **Technical terms and definitions** includes explanation of elementary terms and definitions.
- **Analysis and design** introduces to the topic.
- **Realization** describes and make glosses to a implementation.
- **Experimental measurement** detects a effect of scan-chains in the test compression.
- **Conclusion** is assuming results of the research.

Chapter 1

Technical terms and definitions

In the beginning, I would like to introduce terms and definitions, which are connected with my bachelor thesis. Terms and definitions could be different according to publication because English terminology is not uniform at all. The next terms and definitions are often simplified and taken over from mostly using sources or from presentations of subject MI-TSP (Testing and Reliability) teaching by my supervisor Petr Fišer.

- **Logic gate** is a modeled or physically created component of logic circuits characterized by Boolean function, which has got typically one or more inputs but only one output[3].
- **Latch** or the flip-flop is an elementary storage component of the sequential logic circuit. It has got two stable states, which could be changing depends on signals applied on inputs[3].
- **Digital logic (circuit)** is a logic circuit, where is assigned to every electronic signal to one discrete value. This value represents information, which is processed. In our case, we will be working with circuits, which use the binary representation of signals (”0”, ”1”)[4].
- **Combinational logic (circuit)** is time-independent logic, digital logic, typically characterized by the Boolean logic, where the output is dependent only on the present inputs[4].
- **Sequential logic (circuit)** is a logic circuits, where the output could be depending on the present values of inputs but also on the sequence of the history of the past input[4].
- **Test Pattern** is a testing vector, which assigns value to every input of the circuit[5].
- **Testing** is a verification that produced circuit works as we wanted[2].

- **Defect** is a circuit's wrong issue on the physical level[2].
- **Fault** is a circuits' wrong issue on the logic level[2].
- **Error** is a real demonstration of the wrong issue (wrong numbers addition)[2].
- **Failure** is a real demonstration of the weighty wrong issue (collapse of the computer)[2].
- **Undetectable fault** (redundant, untestable) is a fault, for which does not exist a test pattern[5].

Chapter 2

Analysis and design

This chapter introduces us to testing and reliability and the basic principles necessary to understand the issue of this work. This chapter analyzes elementary approaches to testing combinational and sequential circuits and describes the algorithm of my solution.

Generally, testing of sequential logic is very hard – a long time of test generation, long test, small fault coverage[6]... Success solution seems to convert sequential to combinational logic, which is a part of my study. My solution is focused on sequential circuits but in my analysis and designs part, I am writing some information about the testing combinational logic too.

In my work, I am not working with digital circuits on the hardware level. My program work with combinational and sequential logic designed in BLIF format.

2.1 Testing and Reliability

2.1.1 Combinational logic testing

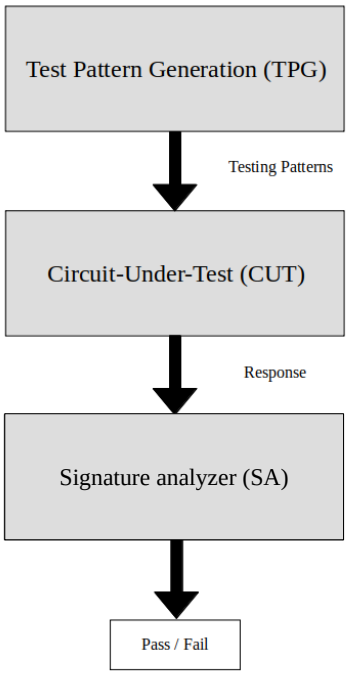

Figure 2.1: Testing of combinational logic[5, 7]

2.1.2 Test Pattern Generation (TPG)

Test Pattern Generator is a generator, which produces test sequences – **test patterns**. We could have deterministic, pseudo-random, or mixed-mode test patterns. Deterministic test patterns are saved in memory (RAM, ROM). These patterns are calculated in advance and they are focused on a given set of faults. Pseudo-random generated patterns produce "random" vectors, which should be able to detect some faults. Mixed-mode testing patterns are a combination of pseudorandom and deterministic test patterns[5].

Test pattern for given fault is an evaluation of primary inputs (PI) that fault is shown in the primary outputs (PO). It does mean that result of the output of the logic circuit is different from the fault loss logic circuit.

Test pattern for given fault must do[5]:

- **Excitation of fault** – pattern should bring inverse value from PI.
- **Propagation of fault** – pattern should display change of fault on the PO.

2.1.3 Test creation

We suppose combinational logic, stuck-at fault model – faults of type stuck-at-1 and stuck-at-0. We want to test every this fault i.e. 100% fault coverage. The question is asked during the creation of logic testing, which patterns are needed to create, and how may test patterns are needed for whole fault coverage of logic. We wanted, of course, to estimate the complexity of this method and find an effective algorithm to solve it.

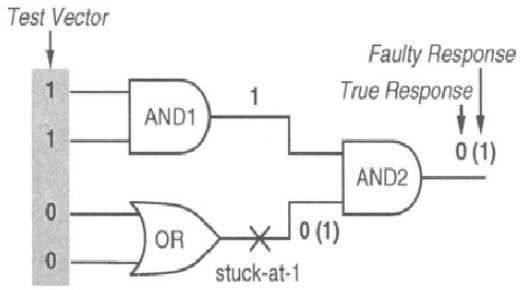

Figure 2.2: Example for single stuck-at-1 fault[8].

$$FaultCoverage = \frac{NumberOfDetectedFaults}{NumberOfAllFaults}$$

We have got two basic types of testing - trivial test and complete test.

- **Trivial test** - is based on applying all vectors on primary inputs. We will apply 2^n vectors, where n is a number of inputs. We have got guaranteed 100% fault static coverage but it is very time-consuming[5].
- **Complete test** - is from definition complete test of 100% fault coverage[5].

2.1.4 Automatic Test Pattern Generation (ATPG)

ATPG is an algorithm tool for generating of test[5]. It finds a test pattern, which enables ATE to recognize working properly logic and fault working logic behavior[5]. ATPG has got two approaches to generate test patterns:

Algorithm 1: Basic ATPG algorithm[9]

Result: Generates test patterns
Input: Combinational logic
Output: Test patterns for the logic

1.Generate fault-list - all fault on the PI, outputs of the logic gates and branches;
2.Reduce fault-list - some faults are equivalent, dominance;
3.Choose the fault - random, first in the list, heuristic...;
4.Generate test pattern;
5.Simulation of test pattern - fault simulation, fault dropping;
6.if *! fault-list.empty* **then**
  repeat **3.**;
7.Compaction of test - remove redundant vectors;

2.1.4.1 Intuitive path sensitization

Intuitive path sensitization is a manual test generation.

Algorithm 2: Intuitive path sensitization algorithm[9]

Result: test vector of PO
1. Choose fault of the logic;
2. signal(fault) = ! signal(fault);
3. Excitation to the PI, determine values of signals because we want to the propagation of fault;
4. From fault, I am proceeding to PO, determine values of signals cause we want to the propagation of fault;
5. I have often got many choices. If the conflict is there, I am returning to 3. and 4.;
6. If I have not got conflicts, in the end, I have got a test vector of PO;
7. If it is not possible to solve the conflict than conflict is not testable;

2.1.4.2 Algorithm approach

ATPG belongs to the NP-complete problems. Detection of test patterns could be reduced on the Boolean satisfiability problem (SAT problem). For some circuits, it is not possible to test all faults. The reason should be a timeout,

backtracking limit (aborted faults), or undetectable faults (redundant faults). Algorithm approaches to solve[9]:

- **Structural** approach is based on the passing through the logic circuit.
- **Algebraic** approach is based on the transfer circuits to Boolean formulas.

Algorithms, which improves of the Intuitive path sensitization algorithm and solve APTG[9]:

- **D-algorithm** - tries to propagate the stuck-at-fault value that reverses value D (for SA0) or not D (for SA1) to a primary output.
- **Path Oriented Decision Making (PODEM)** - is the improvement of the D-algorithm.
- **FAN** - extension of PODEM.
- **Boolean difference**
- **Generation test with SAT** - a high-performance algorithm based on solving the CNF-SAT problem.
- ...

2.1.5 Sequential logic testing

In general terms, we know that testing of sequential logic is demanding on the performance – long time of test generation, long test, weakly fault coverage. Therefore it will be very useful if we will be able to sequential logic transform to combinational logic because we know a relatively simple solution of combinational test generation. Since the 70s, a solution is shown to split the circuit on combinational logic and sequential logic. The solution is simply[6].

On the other hand, we must add new logic to the test. Then circuit has got two modes – **functional** and **testing** mode. This generation model is simple and it brings easy testing of logic. The disadvantage of this solution is the increasing delay, bigger area,... However, many test generators are based on this idea. In my thesis, I am working with one type of model[6].

We have got two basic approachs of tests[6]:

- **Test-per-clock**
- **Test-per-scan**

2.1.5.1 Test-Per-Clock

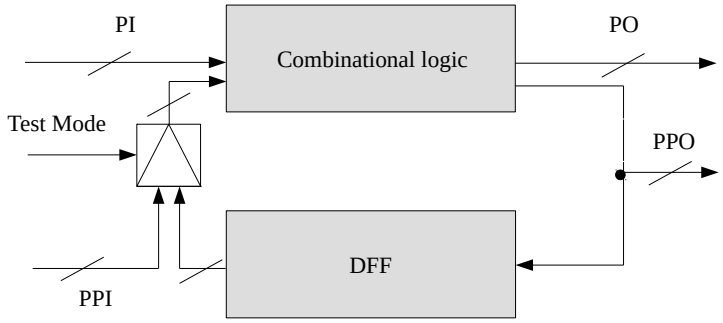

Figure 2.3: Test-Per-Clock approach[6]

Combinational logic is isolated from DFFs. Outputs of DFFs are connected to multiplexor with combinational logic and we are calling them pseudo-primary inputs (PPI). Inputs from DFFs are going to outputs and we are calling them pseudo-primary outputs (PPO). Then inputs of our new circuit are PI + PPI (#DFF) and a number of outputs are PO + PPO (#DFF).

In every cycle, I will be applying one test pattern. It is fast. The disadvantage of this method is that every DFF needs MUX and delays on the way to every DFF. The next disadvantage is the increasing number of inputs and outputs. In practice, it is not useful. It could be used in another method (BIST)[6].

2.1.5.2 Test-Per-Scan

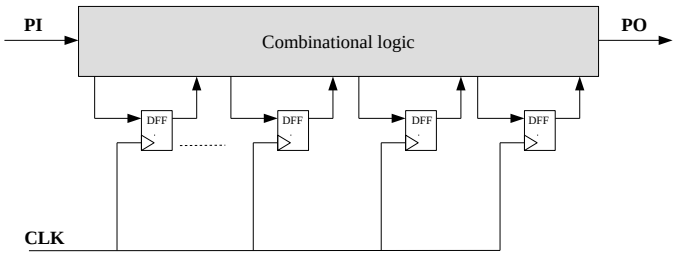

Figure 2.4: Test-Per-Scan approach[6]

Combinational logic is isolated from DFFs. All DFFs are in testing mode connected to one chain called scan-chain. Primary inputs and outputs could be part of the scan-chain. We are adding DFFs to the inputs and outputs. So, we have got #PI + #PO more flip-flops. After that, we have got only one input on a tested circuit and only one output of the tested circuit. It is effective – we have got only three signals in addition – test-mode, scan-in, and scan-out. The main disadvantage of this method is overhead – delay on MUX of every DFFs way, long time of testing, area, data are serial.

Algorithm 3: Test-Per-Scan testing process[6]

Result: test vector of PO (+PPO)
1. Switching to the ”scan” mode.;
2. Putting test pattern to scan-chain.;
3. Switching to functional mode.;
4. Apply one clock cycle. It will be simulating combinational logic. DFFs has got a response.;
5. Taking the response serial, clock cycles like # flip-flops.;
Putting on next pattern.;

Combined and next solutions:

- Partial Scan
- Multiple Scan Chain
- Parallel Scan
- ...

2.1.5.3 Scan-Cell

Scan-Cell based on MUX has got a small increase of area (15-30%), minimum of inputs in addition but it has got delay cause MUX.

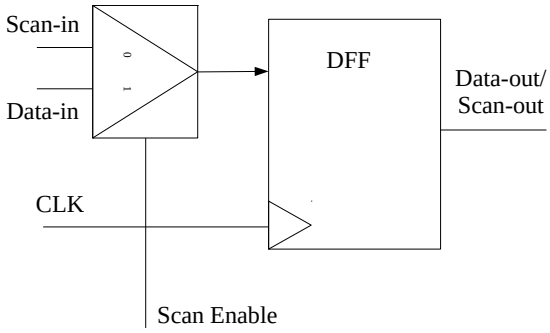

Figure 2.5: Scan-Cell[10]

2.1.6 Cost of Testing

During testing, we are looking at three main variable components.

- **Test Application Time** – Amortizing prize depends on the test time results.
- **Test Data Volume** – Testers has got limited storage.
- **Tester Pins** – In production tester costs depend on inputs pins, which tester supports.

2.1.7 Parallel-Scan

The scan application time should be reduced if we divide the single Scan-chain into more parallel Scan-Chains. The output is consistent. A number of Scan-Chains is depended on the testing tools and devices. Parallel-Scan has got an impact on the compression of test data volume[11].

We could see the principle in Figure 2.6. A combinational compactor is a tree of XOR gates. The output Compactor called Multiple Input Signature Register (MISR), which converts the output to the function called the signature. Next, we compare the output signature with a fault-free signature[11].

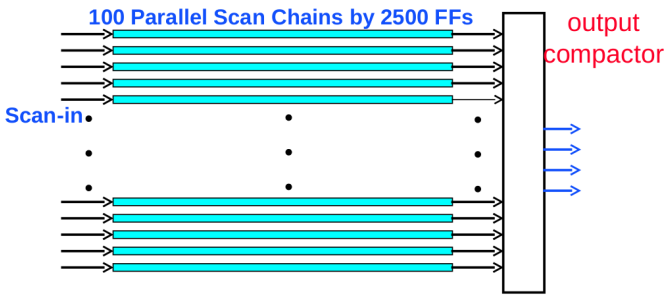

Figure 2.6: Parrallel-Scan approach[1]

2.1.8 Illinois-Scan Architecture

It is a new design for the testability technique Parallel-Serial Full Scan (PSFS) for reducing the complexity of test generation for sequential circuits to combinational circuits test generation problem. Test application time reduction provides dividing the one Scan-chain into multiple parallel scan-chains and we are shifting from the first cells of scan-chains to the last cells of scan-chains if the scan-chain has got some cells[**?**].

This method decreases test data volume too because we have got the parallel connection between every scan-chains cell according to the index of scan-cell and input depends on scan-in, which has got length such a longest scan-chain (for every index of cells I need only one value)[1].

Illinois-Scan Architecture takes advantage of a large circuit, which contains many independent subcircuits. If subcircuits are dependent, it could because of fault coverage loss. Every scan-chain, if it is possible, is parallel scanned with the same scan scan-in input. Scan-ins are usually called scan-in pin in practice. The scan-chains output is directed to a Multiple Input Signature Register (MISR), which calculates signals to scan-out[1, 12] .

We have got two modes of Illinois-Scan:

- **Broadcast Test Mode** - Reduces-Scan time and scan data by a factor #scan-chains. We need more vector to fault coverage and some faults coverage could be reduced[1].

- **Serial Test Mode** - It is using for covering the loss of fault coverage in the Broadcast Mode. It is not often used in industry[**?**].

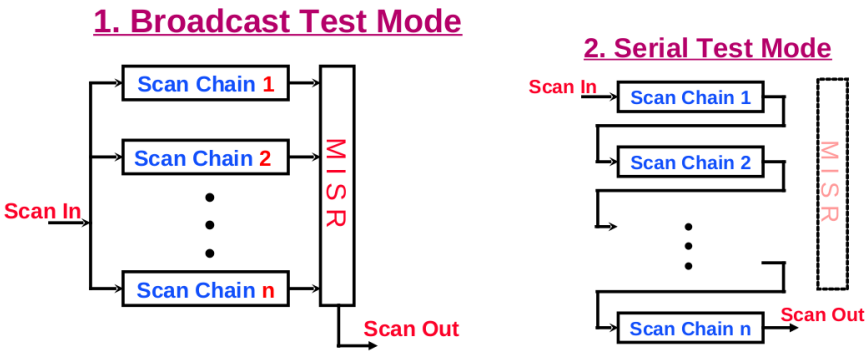

Figure 2.7: Illinois Modes[1]

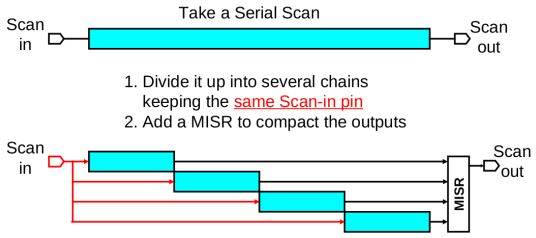

Figure 2.8: One Scan-chain splitted to smaller scan-chains[1]

2.1.8.1 Scan-chains

The first long Scan-chain is created from primary inputs PI and derivated from pseudo-primary inputs PPI created from outputs of flip-flops in arbitrary order. This long Scan-chain is divided on more scan-chains according to our setting. We could randomize them but mostly they are choosing similarly long scan-chains. It reduces test data volume but fault coverage could be decreasing.

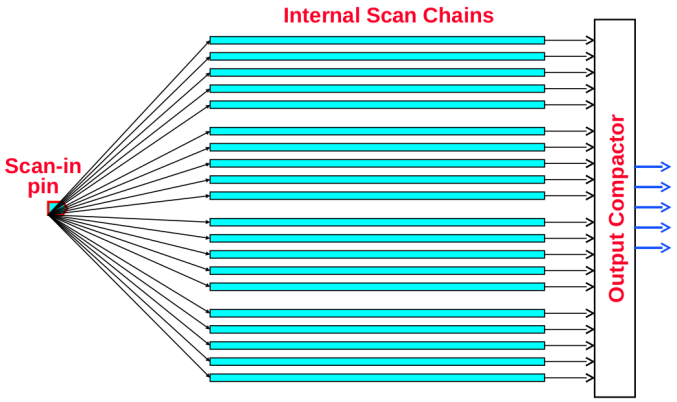

Figure 2.9: Scan-in tester pin connected with scan-chains cells[1]

2.1.8.2 Faults coverage in Illinois-Scan

We could solve tests in a much better time than before cause a parallel scan. On the other hand, we could miss some faults because scan-cells on the same index are connected with scan-in. In the picture below (Figure 3.10) we could see connected scan-cells. You could see that for cells on the same index will be used only 000 or 111. We could not try values 001, 100,... So, we are not able to test these options and it is for us untestable[1].

However, the advantages of this method are indisputably, so this algorithm is still using in practice[1].

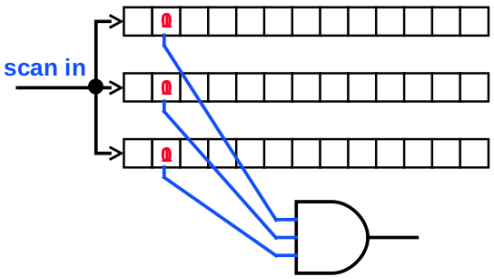

Figure 2.10: Scan-chains faults coverage[1]

2.2 Test Compression based on Illinois-Scan Architecture

Test Compression is a method for reducing the time and cost of testing digital circuits. The Illinois-Scan compression test is trying to reduce bits of our test. For manually generated test patterns is not possible to get good fault coverage in a reasonable time. Therefore, we are using ATPG for test generation. It could reduce test pattern size and if exists, we could test more test patterns then origin circuit. In my thesis, I am using Illinois-Scan Architecture, which reduces input's data volume according to the max size of the biggest scan-chain generated from the tested circuit[11].

Chapter 3

Realization

3.1 Code source

For my bachelor thesis, I wrote three small programs, which are connected with my topic. Programs were written in language C++. More information about implementation and usage read in README.dm enclosed in the attachment. In my text, I will be focused on describing algorithms and structures of implemented models and data types important for my work.

3.2 Data collection

My programs are working with BLIF format standards. This is a wide format standard for describing combinational and sequential logic (model references, subfile references, ...). However, my program is working only with logic gates and latches (DFFs), which are characteristic of the sequential logic. For the illustrative, I would like to show one basic and simple example of a BLIF file, which will be the pattern for the demonstration of algorithms. My experimental part of the thesis will be using BLIF format too.

```
# ------------------------------------
# s27.blif - sequential circuit
.model s27.blif
.inputs G0 G1 G2 G3
.outputs G17
.latch     G10 G5    3
.latch     G11 G6    3
.latch     G13 G7    3
.names G11 G17
0 1
.names G14 G11 G10
00 1
```

```
.names G5 G9 G11
00 1
.names G2 G12 G13
00 1
.names G0 G14
0 1
.names G14 G6 G8
11 1
.names G1 G7 G12
00 1
.names G12 G8 G15
1- 1
-1 1
.names G3 G8 G16
1- 1
-1 1
.names G16 G15 G9
0- 1
-0 1
.end
# ----------------------------------------
```

3.2.1 Simply explanation and characteristic

- **.model** - It is parameter for the logic title[13].
- **.inputs** - It is a group of inputs - names of input signals[**?**].
- **.outputs** - It is a group of outputs - names of output signals[**?**].
- **.latch** - It declares DFF. The first parameter is an input signal, second is an output, the next parameters are not important for this thesis. More you could read in[**?**].
- **.names** - It declares a logic-gate, A logic-gate associates a logic function with signals in the model, which can be used as an input to the other logic functions. The first row declares input signals, the last signal on the row is output. On the next rows are individually separated input signals and output signals. Signal "1" means that signal has got voltage, which is interpreted like a "1". Signal "0" means that signal has got voltage, which is interpreted like a "0". "-" means that signal is not used yet, only could be in input.[**?**].
- **.end** - It is the end of the model (one BLIF could have more models)[**?**].

3.3 Scan-chains generator

It serves for the generating configuration files with configured scan-chains. For serial Illinois-Scan, we are generating only one Scan-chain, where every scan-cells are connected with primary inputs PI (+ pseudo-inputs from the output of origin latch's outputs PPI) and primary outputs PO (pseudo-primary outputs from origin latches inputs PPO). My implemented scan generator has got three modes – it generates scan-chains with random length (set modulo), a number of set scan-chains, and scan-chains according to scan-chain size. Scan-cells order is randomized.

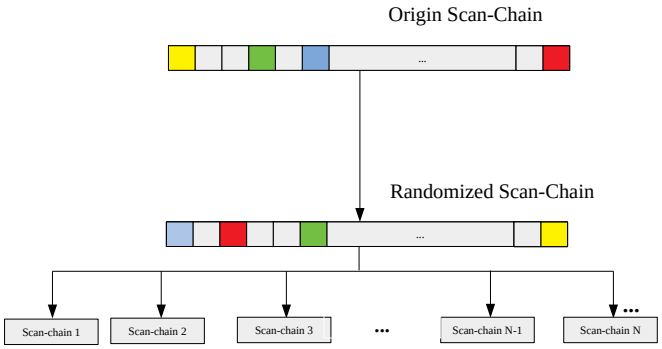

Figure 3.1: Scan-chain splited to scan-chains[14]

3.3.1 Algorithm

Algorithm 4: Generate scan-chains according to size

Result: Configuration of scanchains
Input: BLIF - sequential logic,
A) max size of scan-chain
B) number of scan-chain
C) size of scan-chain
Output: TXT file with scan-chains according to size

---

1. Transform Sequential logic to Combinational logic
- latch's outputs convert to PPI, inputs = PI + PPI)
- latch's inputs convert to PPO, outputs = PO + PPO)

2. Load inputs from Combinational logic (PI + PPI)
- you have got one big SCAN-CHAIN from inputs
- scan-chain is created from scan-cell, scan-cell is represented with names of inputs

3. Randomize of SCAN-CHAIN
- we want to have randomize ordered scan-cells in big scan-chain for testing

4A. Divide SCAN-CHAIN according to scan-chain's randomized size
- divide SCAN-CHAIN to the randomized scan-chain's size
- scan-chains are choosen according randomized size and scan-chain is substracted from the big SCAN-CHAIN until is empty

4B. Divide SCAN-CHAIN according to number of scan-chains
- divide SCAN-CHAIN to the mostly uniformly scan-chains - my method:

count integer size SIZE = size(SCAN-CHAINS) / number
count MODULO = size(SCAN-CHAINS) % number
- for every scan-chain from row 0 to MODULO => scan-chain.size = SIZE + 1
- for every scan-chain from row MODULO + 1 to size(SCAN-CHAINS) => scan-chain.size = SIZE

4C. Divide SCAN-CHAIN according to scan-chain's size
- divide SCAN-CHAIN to the same size scan-chains
- extra scan-cells are saved to the new scan-chain

5. Generate scan-chains to TXT
- generate scan-chains to file and separate them with entering

3.3.2 Usage and examples

3.3.2.1 Scan-chains according to randomize size

We are generating a scan-chains with randomized size. Choose the max size of scan-chains and scan-chains will be generating with this max size.

```
./scan_generator s27.blif -scan_random 3
```

```
G5 G6
G2
G1 G0 G3
G7
```

```
./scan_generator s27.blif -scan_random 4
```

```
G6
G2 G7
G1 G0 G3 G5
```

3.3.2.2 Scan-chains to number of scan-chains

We are generating a number of the most uniformly scan-chains, which could be possible.

```
./scan_generator s27.blif -scan_number 3
```

```
G1 G5 G0
G6 G2
G7 G3
```

```
./scan_generator s27.blif -scan_number 4
```

```
G5 G1
G0 G6
G7 G3
G2
```

3.3.2.3 Scan-chains according to scan-chains size

We are generating scan-chains with assigned size. Extra scan-cells are in the new scan-chain.

```
./scan_chain s27.blif -scan_size 3
```

```
G3 G7 G6
```

G1 G2 G0
G5

```
./scan_chain s27.blif -scan_size 4
```

G5 G1 G3 G6
G7 G2 G0

3.4 Illinois-Scan Architecture

3.4.1 Illinois-Scan Architecture - Full Scan Technique

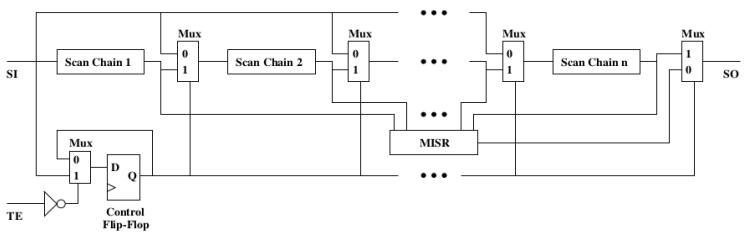

Figure 3.2: Illinois-Scan architecture - Parallel Serial Full Scan (PSFS) Technique[11]

Illinois-Scan Architecture was proposed for the first time as the Parallel-Serial Full Scan (PSFS) Architecture. The algorithm proves in practice that it has got lower test application time compared to any normal full scan architecture. The power of this method is in test data volume reduction too.

The structure of the PSFS technique is shown in Figure 3.2. Scan-chains are constituted by scan-cells. I am using typical scan-cells from Figure 2.5. Every scan-chain is connected with multiplexer with SI (Scan-in pin)[11]. Multiplexer switch between test and normal mode. Originally algorithm was proposed for testing full scan embedded cores. The important advantage of this method is that for each test pattern, the PSFS reduces the tester storage demands and the amount of test data that should be transferred from the tester to the core and from the core back to the tester[12].

My implementation divides origin serial big scan-chain according to configuration and connects scan-chains and scan-in SI (and SO) with the multiplexer. Test mode is not realized with control flip-flop like in Figure 3.2, it is implemented for simplicity like a normal input signal. My implementation does not support randomize output scan-cells and output scan-chain is the same for all variants of Illinois-Scan.

3.4.1.1 Algorithm

Algorithm 5: Illinois-Scan algorithm - Full scan

Result: Generated Illinois-Scan Architecture
Input: BLIF file - sequential circuit, Configuration of scan-chains.(file or could be created)
Output: BLIF file - combinational circuit configured by scan-chains.

---

1. Separate DFFs and Combinational logic
- add new input for testing purpose - TEST
- add new input SI for testing mode
- add new output SO for testing mode
- latch's outputs convert to PPI, inputs = PI + PPI + TEST)
- latch's inputs convert to PPO, outputs = PO + PPO + SO)

2. Create scan-cells from PI, PPI and PO, PPO
- create scan-cells from PI and PPI
- create scan-cells from PO and PPO

3. Create one Scan-chain from scan-cells
- randomize scan-cells
- my implementation does not support PO and PPO scan-cells
- connect scan-cells to one Scan-chain

4. Load configuration
- randomize scan-cells order in Scan-chain
- split Scan-chain to the scan-chains according to chosen options
- you get list of scan-chains

5. Connect schan-chains and SI with multiplexer
6. Generate Header of BLIF
7. Generate scan-chains logic functions with multiplexers
8. Generate Combinational logic from 1.
9. You have got full Illinois-Scan Architecture

3.4.1.2 Example

```
scan.txt: (with input scan-chains)
G2 G0 G3
G7 G1 G6
G5

./illinois_scan s27.blif -scan_name scan.txt
# Header of BLIF
.model N/A
.inputs SC0 SC1 SC2 SI TEST
.outputs OG17 OG10 OG11_EXTRA OG13 SO

# latches of input scan-chains
.latch IG0 OG0
.latch IG1 OG1
.latch IG2 OG2
.latch IG3 OG3
.latch IG5 OG5
.latch IG6 OG6
.latch IG7 OG7

# latches of output scan-chain
.latch IG17 OG17
.latch IG10 OG10
.latch IG11_EXTRA OG11_EXTRA
.latch IG13 OG13


# scan-chain 1
.names SI SC0 TEST IG2
1-0 1
-11 1
.names OG2 G2
1 1
.names G2 SC1 TEST IG0
1-0 1
-11 1
.names OG0 G0
1 1
.names G0 SC2 TEST IG3
1-0 1
-11 1
.names SI OG3 TEST G3
```

```
1-0 1
-11 1

# scan-chain 2
.names SI SC0 TEST IG7
1-0 1
-11 1
.names OG7 G7
1 1
.names G7 SC1 TEST IG1
1-0 1
-11 1
.names OG1 G1
1 1
.names G1 SC2 TEST IG6
1-0 1
-11 1
.names SI OG6 TEST G6
1-0 1
-11 1

# scan-chain 3
.names SI SC0 TEST IG5
1-0 1
-11 1
.names SI OG5 TEST G5
1-0 1
-11 1

# scan-chain 4 - output scan-chain
.names G17 OG10 TEST IG17
1-0 1
-11 1
.names G10 OG11_EXTRA TEST IG10
1-0 1
-11 1
.names G11_EXTRA OG13 TEST IG11_EXTRA
1-0 1
-11 1
.names G13 G5 TEST IG13
1-0 1
-11 1
.names OG17 G17 TEST SO
1-1 1
```

```
-01 1

# combinational logic
.names G14 G11 G10
00 1
.names G5 G9 G11
00 1
.names G11 G11_EXTRA
1 1
.names G1 G7 G12
00 1
.names G2 G12 G13
00 1
.names G0 G14
0 1
.names G12 G8 G15
00 1
.names G3 G8 G16
00 1
.names G11 G17
0 1
.names G14 G6 G8
11 1
.names G16 G15 G9
11 1
.end
```

3.4.2 Illinois-Scan Architecture Generation - Reduced for ATPG testing purpose

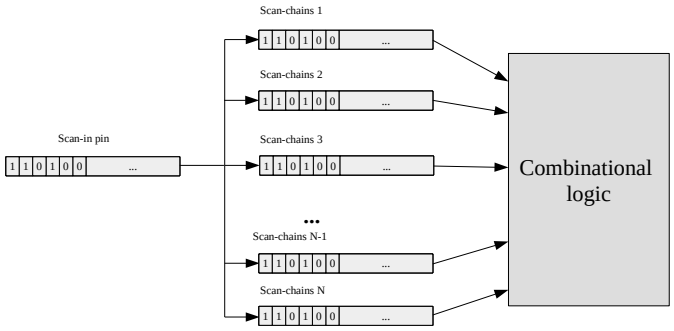

Figure 3.3: Reduced Illinois-scan architecture using for our ATPG testing[15]

For our ATPG testing purpose, we reduced architecture only on combinational logic and scan-chains, which are connected with the circuit's PI and PPI. Outputs are PO and PPO for our new architecture. We throw all latches and create from origin inputs of latches pseudo-primary outputs PPO and from outputs of latches we create new pseudo-primary inputs PPI. Scan-cells are created simply from input signals and output signals, which are not changed during the algorithm. Origin combinational logic is not changed too.

3.4.3 Model stucture

In my thesis, I am using a class model like an internal representation of the BLIF file. It represents inputs, outputs, all logic gates, and their connections and other basic information from BLIF. It does print a logic gates connection.

```
# s27.blif model structure printing
# inputs are in [] brackets
# outputs are in () brackets
# N/A is the model title

N/A+-(G17)-G11+-[G5]
    |              \-G9+-G16+-[G3]
    |                    |     \-G8+-G14-[G0]
    |                    |           \-[G6]
    |                    \-G15+-G12+-[G1]
    |                          |     \-[G7]
    |                          \-G8
    |-(G10)+-G14
    |       \-G11
    |-(G11_EXTRA)-G11
    \-(G13)+-[G2]
            \-G12
```

3.4.4 Configuration

Illinois scan is configured with a list of scan-chains. We could generate a list with illinois_scan parameters (scan_generator) or we could create manually the list of scan-chains. The configuration file has got on every line one scan-chains, which is a list of logic gates.

3.4.4.1 Examples

```
./scan_generator ../samples/s27.blif -scan_size 4
G0 G2 G5 G1
G3 G6 G7

./scan_generator ../samples/s27.blif -scan_number 4
G1 G5
G6 G2
G7 G3
G0

./scan_generator ../samples/s27.blif -scan_random 4
G0
```

G5 G2
G6
G1 G7 G3

3.4.5 Algorithm

Algorithm 6: Illinois-Scan algorithm

Result: Generated Illinois-Scan Architecture
Input: BLIF file - sequential circuit, Configuration of scan-chains.(file or could be created)
Output: BLIF file - combinational circuit configured by scan-chains.

---

1. Transform Sequential logic to Combinational logic
- latch's outputs convert to PPI, inputs = PI + PPI)
- latch's inputs convert to PPO, outputs = PO + PPO)

2. Load configuration
- create your own or generate from scan_generator (or use illinois_scan parameters)
- you get list of scan-chains

3. Create new inputs
- new inputs will be indexes of scan-chains #new_inputs_number = N = #max(size of scan-chain from loaded scan-chains)

=> we have got new inputs = SC0, SC1, SC2, ... SCN-1 (SC - scan-cell)
- outputs are unchanged

4. Generate header of Illinois-scan architecture
- generates title, inputs, outputs...

5. Generate scan-chain logic gate function
- generates for every scan-chain cell new logic gate function according to index
- every input SC$i is connected with every scan-chain[$i] from scan-chains and they are connected with the same value i.e.:

```
for scan-chain from scan-chains do
    for i=0; i¡scan-chain.size(); ++i do
        .names SC$i scan-chain[$i]
        0 1
    end
end
```

6. Generate Combinational logic from 1.
- generate same combinational logic, which was transformed in step **1.**

7. Illinois Scan Architecture
- You generated Illinois Scan Architecture according to configuration
- test compression is depended on scan-chains - the size of the test pattern will be the max size of scan-chain from origin scan-chains configuration

3.4.6 Usage and examples

```
in scan.txt:
G1 G6
G3 G0 G5
G7 G2

./illinois_scan ../samples/s27.blif -scan_name scan.txt
.model N/A
.inputs SC0 SC1 SC2
.outputs G17 G10 G11_EXTRA G13

.names SC0 G1
1 1
.names SC1 G6
1 1
.names SC0 G3
1 1
.names SC1 G0
1 1
.names SC2 G5
1 1
.names SC0 G7
1 1
.names SC1 G2
1 1

.names G14 G11 G10
00 1
.names G5 G9 G11
00 1
.names G11 G11_EXTRA
1 1
.names G1 G7 G12
00 1
.names G2 G12 G13
00 1
.names G0 G14
0 1
.names G12 G8 G15
00 1
.names G3 G8 G16
00 1
.names G11 G17
0 1
.names G14 G6 G8
```

```
11 1
.names G16 G15 G9
11 1
.end
```

Chapter 4

Experimental measurement

4.1 Introduction

Finally, we are going to the experimental part of my thesis. As I said in the previous capture, experimental data comes from a reduced Illinois-Scan algorithm and you could see measured data in the appendix B - Experimental data. Our generated files based on Illinois-Scan Architecture was measured on the ATPG Atalanta. This ATPG is a modified ATPG tool and fault simulator, which was developed by Virginia Tech University[16]. At the beginning of the experimental chapter, I would like to explain data from the tables in the appendix. I am proceeding from ATPG Atalanta manual desctiption[16]:

- **gates** - It means a number of CUT gates in the circuit.
- **iv** - It means a number of CUT primary inputs.
- **ov** - It means a number of CUT primary outputs.
- **i_patterns** - It is a number of test patterns before compaction (or final, if no compaction).
- **patterns** - It is a number final test patterns (after compaction).
- **faults** - It is a number of processed faults.
- **d_faults** – It is a number of detected faults.
- **r_faults** – It is a number of identified redundant faults.
- **time** – It is a CPU time [s].
- **FC** – It is a fault coverage [
- **#scan-chains** – It is a number of scan-chains in our circuit.

4.2 Dependence fault coverage and scan-chains

In the first test set, I was analyzing dependence faults coverage on the number of scan-chains. In the second test set, I was analyzing dependence faults coverage on the randomized scan-chains cell with the same length. I took origin combinational circuits such a model and pattern, which was compared with the results of measurement circuits based on Illinois-Scan architecture. If you look at the columns called fault, d_fault, r_fault in the tables, you could see that detected faults are in most cases lower. It is important to realize, that not every fault could be detected (or it is hard to measure them in a reasonable time). This is a reason, why fault coverage is measured from detected faults, not from total faults on the circuit. I am calculating fault coverage from the following equation:

$$FC = \frac{DetectedFaultsOnCircuit}{DetectedFaultsInOriginCircuit}$$

4.2.1 Number of scan-chains

I chose selected numbers from 2 until 100. We could see the results of the measurement in Table B.2. In Figure 4.1, which is generated from Table B.2, we could see that an amount of scan-chains has got the most impact on the smaller circuits. You could see that smaller Illinois-Scan dropped on the 5% caused an increase in the number of scan-chains. It is caused by cutting options of the original inputs. If inputs are dependent on combinational logic, it will be shown on the fault coverage loss.

On the other hand, we could see that fault coverage of big circuits is descending slowly compared to smaller files. In bigger circuit is less chance that randomized inputs are dependent on logic, fault coverage will be higher. You could see in Figure 4.1 that the fault coverage of the circuit is still more than 80% after dividing inputs to twenty scan-chains but the test data volume is much less. The test pattern is twenty smaller than the original test pattern.

So, a number of scan-chains are considerable for smaller circuits and we could have marked fault coverage loss. For big circuits, it has not got marked fault coverage but the size of test patterns is very reduced.

4.2.2 Randomized scan-cells in scan-chains

In the second test set of this part, I chose ten randomized scan configurations for every circuit and with the same number of scan-chains. I chose number 10. We could see data from measurement in Table B.3. Figure 4.2 proceeds from this table. From the chart, we could see that randomized scan-cells have not got a significant impact on the results of the measurement.

So, randomizing scan-cells has not got a significant impact on our measurement. We could see it in the charts on the next page.

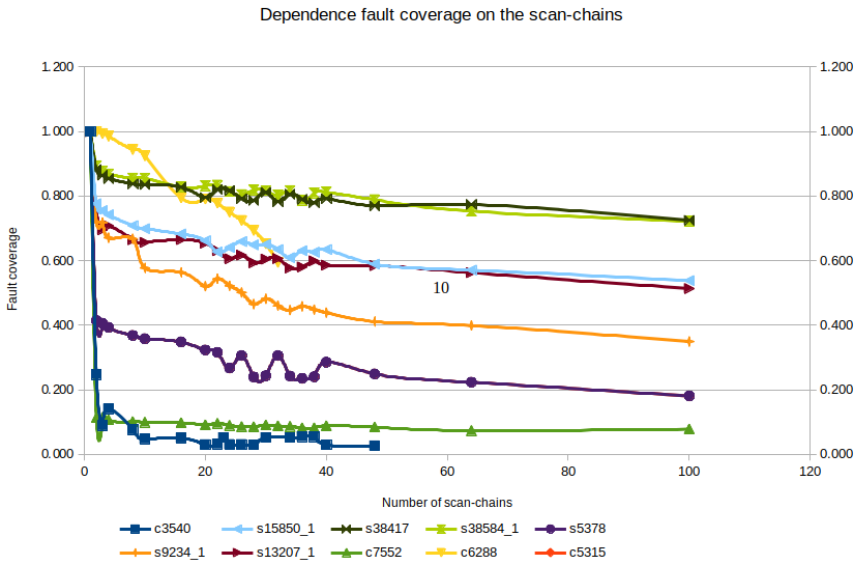

Figure 4.1: Dependence faul coverage on the scan-chains[17]

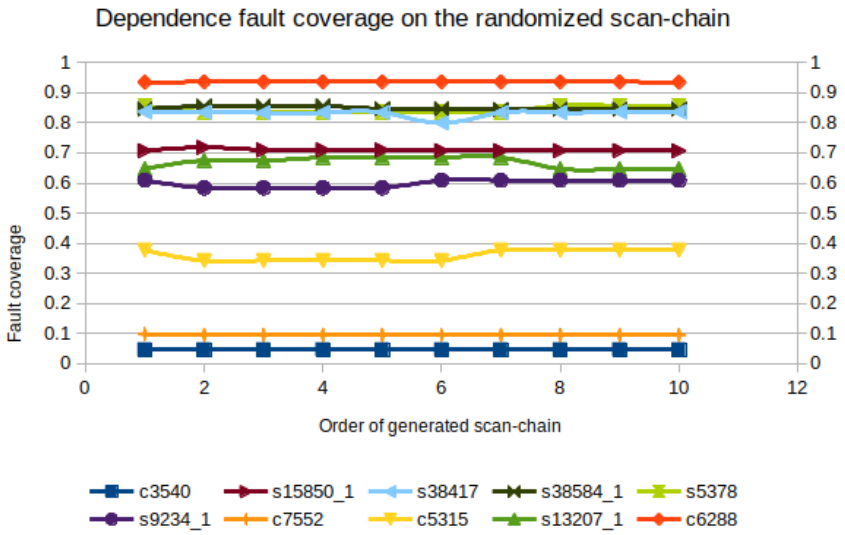

Figure 4.2: Dependence fault coverage on the randomized scan-chains[18]

4.3 Dependence test patterns and scan-chains

In this part, we are finding the dependencies between a number of generated test patterns and scan-chains. Similarly, as the previous page, the number of the test pattern depends on the number of scan-chains

4.3.1 Number of scan-chains

In the beginning, I would like to explain that test patterns generated from the Illinois-Scan are smaller than the original logic. It is caused by Illinois scan architecture. During Illinois-Scan architecture, we have got compressed inputs according to the size of scan-chains. Fewer inputs, we need smaller test pattern. On the other side, at the same time, according to studdies, we should generate more test patterns because we get more time for the testing and saved data volume. This hypothesis did not confirm on our experimental data. When I was finding the root of this problem I find out that cause could be in generating test patterns [11]. If we have got relatively small data volume, our biggest circuits have got more than 22 thousand of logic gates (chips have got hundreds of thousands), it is possible to calculate it in a reasonable time. For hundreds of thousands, it will be hard to count it.

So, in Figure 4.3, we could see similarly distribution of data like in Figure 4.1.

4.3.2 Randomized scanchains

In this part, I was finding dependencies between a number of the test patterns and randomized scan-cells in scan-chains with the same number of scan-chains. We could see in Figure 4.4. that some configuration of scan-chains gets changed a number of test patterns. If we compare it with Figure 4.2, we find out that randomized scan-chains could have a bigger effect on test patterns than fault coverage.

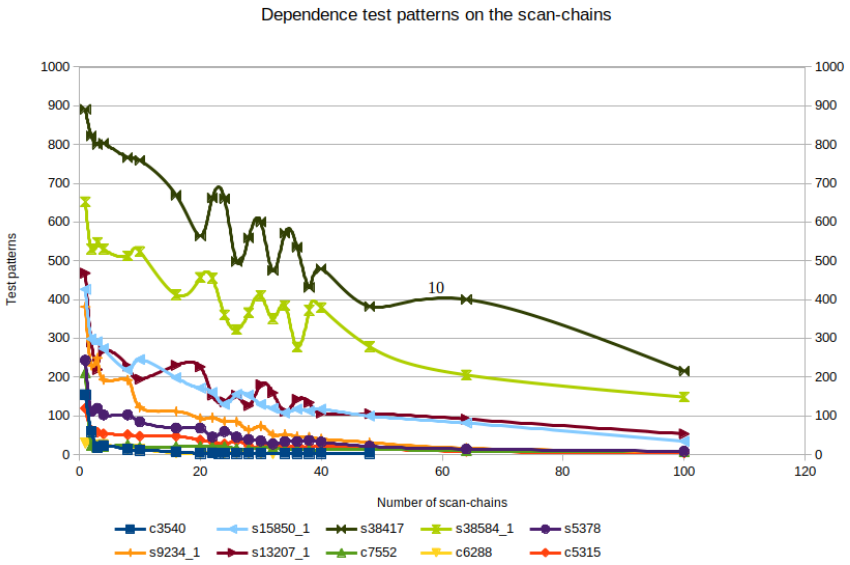

Figure 4.3: Dependence generated test patterns on the scan-chains[19]

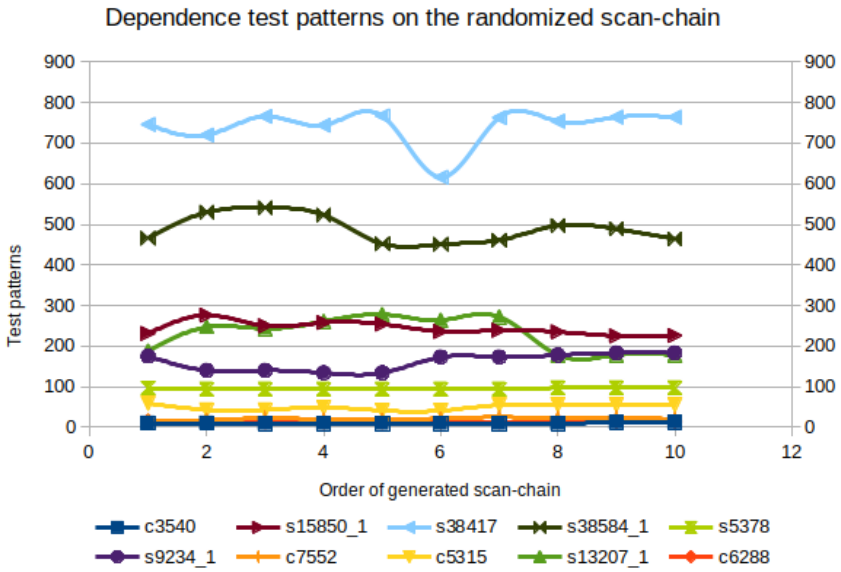

Figure 4.4: Dependence generated test patterns on the randomized scan-chains[20]

4.4 Dependence fault coverage and test patterns

For illustration, I decided to show the dependence between fault coverage and the number of test patterns. According to studies[1], test data volume has got an effect on fault coverage. If you will try every existing test pattern, you get 100% coverage of detectable faults. As you see in Figure 4.5, this prediction was fulfilled on the experimental measurement.

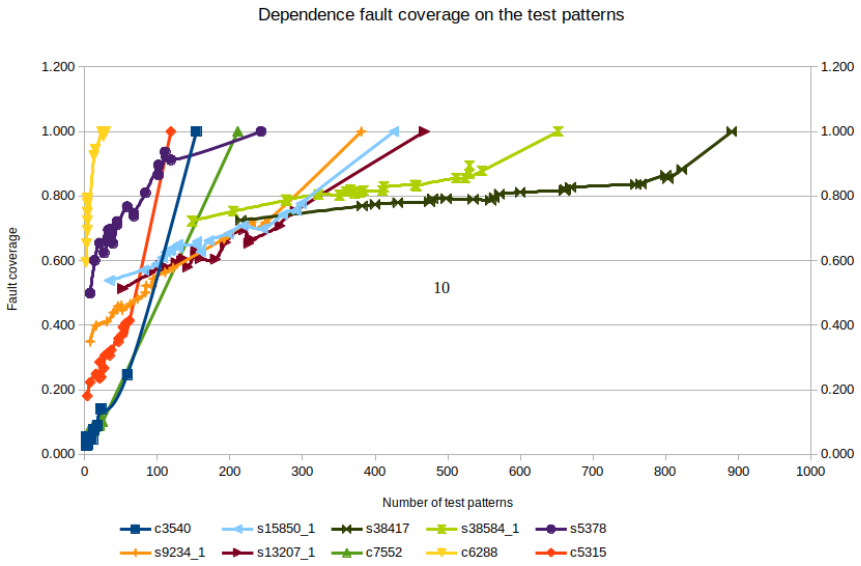

Figure 4.5: Dependence generated test patterns on the randomized scan-chains[21]

Conclusion

My bachelor thesis had two main aims. The first aim was to implement and describe a Test compression algorithm based on Illinois-Scan Architecture. The algorithm was described in the bachelor thesis and implemented in programming language C++. The required theory for my thesis was described in the first chapter.

The second aim of my thesis was testing generated Illinois-Scan Architecture on ATPG. Illinois-Scan Architecture was generated according to the configuration of scan-chains. For this purpose, was written Scan Generator, which generates configuration files according to our requirements. My program, which generates Illinois-Scan Architecture, could be generated like a full-scan architecture PSFS or it could be generated in reduced form for purpose of ATPG testing. The experimental part was evaluated in the Experimental measurement chapter.

We could see from experimental measurement that fault coverage of circuits could be rapidly changed according to scan-chain configuration. From experimental measurement, we could see too that this algorithm matches better to extensively circuits. My implementation used only one Scan-in pin. So, I can see the possibility of my research that it could be extended by more Scan-in pins, and I believe that fault coverage of every measured logic could be better.

Bibliography

[1] PATEL, Janak H. Illinois Scan Architecture[lecture]. Urbana-Champaign: Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign

[2] FIŠER, Petr. MI-TSP 1 - Úvod, typy defektů, poruch, testování[lecture]. Prague: FIT CTU. Faculty of information technology, 25.9.2019

[3] JAEGER, Richard C. Microelectronic Circuit Design, McGraw-Hill 1997, ISBN 0-07-032482-4, pp. 226–233

[4] ALEXANDER, H. MATTHEW, S. Fundamentals of Electric Circuits. [s.l.]: McGraw-Hill, 2004, p.15-21

[5] FIŠER, Petr. MI-TSP 2 - Generování testu pro kombinační obvody"[lecture]. Prague: FIT CTU. Faculty of information technology, 2.10.2019

[6] FIŠER, Petr. MI-TSP 8 - Testování sekvenčních obvodů - scan návrh"[lecture]. Prague: FIT CTU. Faculty of information technology, 20.11.2019

[7] KRÁLÍK, Daniel. Test generation picture in src/tehsis/FITthesis-template/TestGeneration.pdf. Prague: FIT CTU, Faculty of information technology, 15.7.2020; FIŠER

[8] RAJ THILAK, K. GAYATHRI, S. Fault coverage analysis using fault model and functional testing for DPM reduction, https://www.semanticscholar.org/paper/Fault-coverage-analysis-using-fault-model-and-for-Thilak-Gayathri/4c4c3d37f929d90452b8b001bfbdc23868c19ccb, 2015

[9] FIŠER, Petr. MI-TSP 3 - Algoritmy pro automatické generování testu (ATPG)[lecture]. Prague: FIT CTU. Faculty of information technology, 9.10.2019

[10] KRÁLÍK, Daniel. Scan-cell in /src/tehsis/FITthesis-template/scan-cell.pdf. Prague: FIT CTU, Faculty of information technology, 15.7.2020

[11] HAMZAOGLU, Ilker & PATEL, Janak. (1999). Reducing Test Application Time for Full Scan Embedded Cores. F. F. Hsu, K. M. Butler and J. H. Patel, "A case study on the implementation of the Illinois Scan Architecture," Proceedings International Test Conference 2001 (Cat. No.01CH37260), Baltimore, MD, USA, 2001, pp. 538-547, doi: 10.1109/TEST.2001.966672

[12] THAKKAR, Chintan S., ANISH T. ILLINOIS SCAN ARCHITECTURE DESIGN. Kharagpur: Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Here: https://pdfs.semanticscholar.org/ad8e/c96c099806369df78a6324ef5341adfcc9db.pdf, p.2-8

[13] ROSS, M. Berkeley Logic Interchange Format (BLIF). Berkeley: Universita of California, Universita of California, 22.2.2005, p.1-5

[14] KRÁLÍK, Daniel. Splitted scan-chain in /src/tehsis/FITthesis-template/scan-chain-split.pdf. Prague: FIT CTU, Faculty of information technology, 15.6.2020

[15] KRÁLÍK, Daniel. Reduced Illinois Architecture in /src/tehsis/FITthesis-template/illinois-architecture-full.pdf. Prague: FIT CTU, Faculty of information technology, 15.6.2020

[16] LEE, H.K. HA, D.S. "Atalanta: an Efficient ATPG for Combinational Circuits,", Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1993. (modified by FIŠER, Petr. September 2005)

[17] KRÁLÍK, Daniel. Chart in /src/tehsis/FITthesis-template/chart-fc-number-sc.png. Prague: FIT CTU, Faculty of information technology, 20.7.2020

[18] KRÁLÍK, Daniel. Chart in /src/tehsis/FITthesis-template/chart-fc-randomized-sc.png. Prague: FIT CTU, Faculty of information technology, 20.7.2020

[19] KRÁLÍK, Daniel. Chart in /src/tehsis/FITthesis-template/chart-tp-number-sc.png. Prague: FIT CTU, Faculty of information technology, 20.7.2020

[20] KRÁLÍK, Daniel. Chart in /src/tehsis/FITthesis-template/chart-tp-randomized-sc.png. Prague: FIT CTU, Faculty of information technology, 20.7.2020

[21] KRÁLÍK, Daniel. Chart in /src/tehsis/FITthesis-template/chart-fc-tp.png. Prague: FIT CTU, Faculty of information technology, 20.7.2020

[22] F. Brglez, H. Fujiwara, A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran, Proceedings of the IEEE International Symposium Circuits and Systems (ISCAS), IEEE Press, Piscataway, N.J., 1985, pp. 677–692.

[23] F. Brglez, D. Bryan, K. Kozminski, Combinational profiles of sequential benchmark circuits, IEEE International Symposium on Circuits and Systems (ISCAS'89), vol.3, 1989, pp. 1929–1934

[24] KRÁLÍK, Daniel. Experimental data in /src/wbdcm/samples/stat/Experimental.ods. Prague: FIT CTU, Faculty of information technology, 20.7.2020

Appendix A

Acronyms

ATPG Automatic Test Pattern Generation / Generator

TPG Test Pattern Generation / Generator

BLIF Berkeley Logic Interchange Format

ATE Automatic Test Equipment

MUX Multiplexer

PI Primary Inputs

PPI Pseudo-Primary Inputs

PO Primary outputs

PPO Pseudo-Primary Outputs

Appendix B

Experimental data

Benchmarks for experimental measurement of my bachelor thesis are on this references[22][23]. This tables were created from experimental data[24].

B.1 Origin combinational logic

In this section are measured data from ATPG tested on the combinational logic.

file	gates	iv	ov	patterns	faults	d_faults	r_faults	time	FC
c3540.rep	1669	50	22	154	3428	3291	137	0.052	1
c5315.rep	2307	178	123	119	5350	5291	59	0.056	1
c6288.rep	2416	32	32	29	7744	7720	34	0.115	1
c7552.rep	3512	207	108	211	7550	7417	71	0.238	1
s13207_1.rep	7979	700	790	468	9815	9664	142	0.398	1
s15850_1.rep	9775	611	684	426	11725	11336	380	0.563	1
s27.rep	11	7	4	8	32	32	0	0	1
s38417.rep	22257	1664	1742	891	31180	31015	161	5.501	1
s38584_1.rep	19405	1464	1730	652	36303	34797	1482	3.353	1
s5378.rep	2836	214	213	243	4551	4511	40	0.091	1
s9234_1.rep	5597	247	250	381	6927	6475	404	0.455	1

Table B.1

B.2 Illinois scan - different number of scan-chains

In this section are measured data from ATPG tested on the Illinois-scan architecture with different number of scan-chains. Scan-chains are generated consistent size.

file	gates	iv	ov	patterns	faults	d_faults	r_faults	time	FC	#scan-chains
c3540-sc10.rep	1719	5	22	11	3438	157	3279	0.341	0.048	10
c3540-sc16.rep	1719	4	22	7	3436	166	3269	0.323	0.050	16
c3540-sc20.rep	1719	3	22	3	3434	99	3335	0.24	0.030	20
c3540-sc22.rep	1719	3	22	4	3434	104	3330	0.277	0.032	22
c3540-sc24.rep	1719	3	22	3	3434	100	3334	0.243	0.030	24
c3540-sc26.rep	1719	2	22	3	3432	100	3332	0.243	0.030	26
c3540-sc28.rep	1719	2	22	3	3432	97	3335	0.229	0.029	28
c3540-sc2.rep	1719	25	22	59	3478	814	2436	0.467	0.247	2
c3540-sc30.rep	1719	2	22	3	3432	171	3261	0.241	0.052	30
c3540-sc32.rep	1719	2	22	3	3432	171	3261	0.223	0.052	23
c3540-sc34.rep	1719	2	22	3	3432	171	3261	0.227	0.052	34
c3540-sc36.rep	1719	2	22	3	3432	180	3252	0.221	0.055	36
c3540-sc38.rep	1719	2	22	3	3432	180	3252	0.232	0.055	38
c3540-sc3.rep	1719	17	22	18	3462	293	2741	0.664	0.089	3
c3540-sc40.rep	1719	2	22	3	3432	97	3335	0.257	0.029	40
c3540-sc48.rep	1719	2	22	3	3432	87	3345	0.239	0.026	48
c3540-sc4.rep	1719	13	22	23	3454	463	2807	0.481	0.141	4
c3540-sc8.rep	1719	7	22	13	3442	251	3177	0.311	0.076	8
c5315-sc100.rep	2485	2	123	4	5354	957	4397	0.404	0.181	100
c5315-sc10.rep	2485	18	123	47	5386	1896	3380	0.458	0.358	10

c5315-sc16.rep	2485	12	123	47	5374	1843	3486	0.428	0.348	16
c5315-sc20.rep	2485	9	123	37	5368	1710	3628	0.442	0.323	20
c5315-sc22.rep	2485	9	123	33	5368	1670	3669	0.46	0.316	22
c5315-sc24.rep	2485	8	123	27	5366	1414	3916	0.501	0.267	24
c5315-sc26.rep	2485	7	123	35	5364	1619	3725	0.445	0.306	26
c5315-sc28.rep	2485	7	123	23	5364	1269	4074	0.502	0.240	28
c5315-sc2.rep	2485	89	123	62	5528	2192	3157	0.42	0.414	2
c5315-sc30.rep	2485	6	123	20	5362	1288	4060	0.486	0.243	30
c5315-sc32.rep	2485	6	123	28	5362	1619	3737	0.408	0.306	32
c5315-sc34.rep	2485	6	123	20	5362	1284	4072	0.442	0.243	34
c5315-sc36.rep	2485	5	123	21	5360	1244	4107	0.444	0.235	36
c5315-sc38.rep	2485	5	123	18	5360	1272	4082	0.412	0.240	38
c5315-sc3.rep	2485	60	123	57	5468	2148	3162	0.404	0.406	3
c5315-sc40.rep	2485	5	123	21	5360	1511	3843	0.4	0.286	40
c5315-sc48.rep	2485	4	123	16	5358	1319	4037	0.398	0.249	48
c5315-sc4.rep	2485	45	123	53	5440	2082	3207	0.408	0.393	4
c5315-sc64.rep	2485	3	123	8	5356	1182	4173	0.375	0.223	64
c5315-sc8.rep	2485	23	123	51	5396	1949	3345	0.438	0.368	8

c6288-sc10.rep	2448	4	32	13	7752	7139	437	0.327	0.925	10
c6288-sc16.rep	2448	2	32	4	7748	6134	1105	0.706	0.795	16
c6288-sc20.rep	2448	2	32	4	7748	6107	1078	0.737	0.791	20
c6288-sc22.rep	2448	2	32	4	7748	6010	1221	0.781	0.778	22
c6288-sc24.rep	2448	2	32	4	7748	5795	1349	0.864	0.751	24
c6288-sc26.rep	2448	2	32	4	7748	5593	1531	1.003	0.724	26

c6288-sc28.rep	2448	2	32	4	7748	5356	1808	1.106	0.694	28
c6288-sc2.rep	2448	16	32	24	7776	7720	56	0.082	1.000	2
c6288-sc30.rep	2448	2	32	3	7748	5037	2286	1.298	0.652	30
c6288-sc32.rep	2448	1	32	2	7746	4592	2837	1.457	0.595	32
c6288-sc3.rep	2448	11	32	25	7766	7671	95	0.085	0.994	3
c6288-sc4.rep	2448	8	32	26	7760	7608	148	0.1	0.985	4
c6288-sc8.rep	2448	4	32	15	7752	7290	320	0.215	0.944	8

c7552-sc100.rep	3719	3	108	7	7556	583	6973	0.903	0.079	100
c7552-sc10.rep	3719	21	108	19	7592	740	6707	1.313	0.100	10
c7552-sc16.rep	3719	13	108	20	7576	726	6784	1.243	0.098	16
c7552-sc20.rep	3719	11	108	21	7572	680	6860	1.179	0.092	20
c7552-sc22.rep	3719	10	108	20	7570	713	6819	1.206	0.096	22
c7552-sc24.rep	3719	9	108	20	7568	668	6866	1.231	0.090	24
c7552-sc26.rep	3719	8	108	15	7566	638	6914	1.165	0.086	26
c7552-sc28.rep	3719	8	108	14	7566	635	6924	1.099	0.086	28
c7552-sc2.rep	3719	104	108	22	7756	841	6640	1.383	0.113	2
c7552-sc30.rep	3719	7	108	17	7564	675	6880	1.068	0.091	30
c7552-sc32.rep	3719	7	108	17	7564	656	6900	1.065	0.088	32
c7552-sc34.rep	3719	7	108	13	7564	644	6912	1.1	0.087	34
c7552-sc36.rep	3719	6	108	13	7562	610	6946	1.115	0.082	36
c7552-sc38.rep	3719	6	108	12	7562	610	6950	1.078	0.082	38
c7552-sc3.rep	3719	69	108	21	7688	809	6638	1.403	0.109	3
c7552-sc40.rep	3719	6	108	12	7562	658	6894	1.032	0.089	40
c7552-sc48.rep	3719	5	108	14	7560	626	6931	0.943	0.084	48

c7552-sc4.rep	3719	52	108	21	7654	795	6655	1.316	0.107	4
c7552-sc64.rep	3719	4	108	9	7558	547	7009	0.916	0.074	64
c7552-sc8.rep	3719	26	108	25	7602	753	6694	1.269	0.102	8

s13207_1-sc100.rep	8679	7	790	53	9829	4964	4864	1.073	0.514	100
s13207_1-sc10.rep	8679	70	790	194	9955	6347	3586	0.873	0.657	10
s13207_1-sc16.rep	8679	44	790	230	9903	6431	3453	0.911	0.665	16
s13207_1-sc20.rep	8679	35	790	226	9885	6323	3540	0.964	0.654	20
s13207_1-sc22.rep	8679	32	790	153	9879	6097	3768	0.935	0.631	22
s13207_1-sc24.rep	8679	30	790	136	9875	5856	4002	0.995	0.606	24
s13207_1-sc26.rep	8679	27	790	153	9869	5966	3894	0.918	0.617	26
s13207_1-sc28.rep	8679	25	790	126	9865	5736	4118	1.103	0.594	28
s13207_1-sc2.rep	8679	350	790	290	10515	7276	3210	0.835	0.753	2
s13207_1-sc30.rep	8679	24	790	180	9863	5845	3999	1.097	0.605	30
s13207_1-sc32.rep	8679	22	790	159	9859	5864	3969	1.051	0.607	32
s13207_1-sc34.rep	8679	21	790	111	9857	5580	4265	0.963	0.577	34
s13207_1-sc36.rep	8679	20	790	142	9855	5611	4243	0.941	0.581	36
s13207_1-sc38.rep	8679	19	790	134	9853	5794	4047	1.14	0.600	38
s13207_1-sc3.rep	8679	234	790	219	10281	6724	3537	0.776	0.696	3
s13207_1-sc40.rep	8679	18	790	107	9851	5666	4175	0.907	0.586	40
s13207_1-sc48.rep	8679	15	790	105	9845	5652	4192	0.892	0.585	48
s13207_1-sc4.rep	8679	175	790	269	10165	6845	3289	0.835	0.708	4
s13207_1-sc64.rep	8679	11	790	91	9837	5441	4396	1.003	0.563	64
s13207_1-sc8.rep	8679	88	790	228	9991	6427	3533	0.843	0.665	8

s15850_1-sc100.rep	10386	7	684	34	11739	6103	5625	2.252	0.538	100
s15850_1-sc10.rep	10386	62	684	245	11847	7925	3819	1.453	0.699	10
s15850_1-sc16.rep	10386	39	684	198	11803	7743	4007	1.365	0.683	16
s15850_1-sc20.rep	10386	31	684	171	11787	7501	4211	1.57	0.662	20
s15850_1-sc22.rep	10386	28	684	160	11781	7112	4546	2.254	0.627	22
s15850_1-sc24.rep	10386	26	684	129	11777	7266	4404	1.833	0.641	24
s15850_1-sc26.rep	10386	24	684	155	11773	7473	4194	1.877	0.659	26
s15850_1-sc28.rep	10386	22	684	153	11769	7357	4319	1.643	0.649	28
s15850_1-sc2.rep	10386	306	684	298	12335	8804	3456	1.231	0.777	2
s15850_1-sc30.rep	10386	21	684	129	11767	7373	4326	1.713	0.650	30
s15850_1-sc32.rep	10386	20	684	119	11765	7193	4485	1.892	0.635	32
s15850_1-sc34.rep	10386	18	684	107	11761	6908	4753	2.06	0.609	34
s15850_1-sc36.rep	10386	17	684	117	11759	7153	4578	1.582	0.631	36
s15850_1-sc38.rep	10386	17	684	112	11759	7099	4611	2.124	0.626	38
s15850_1-sc3.rep	10386	204	684	291	12133	8562	3496	1.241	0.755	3
s15850_1-sc40.rep	10386	16	684	117	11757	7193	4514	2.056	0.635	40
s15850_1-sc48.rep	10386	13	684	99	11751	6682	4968	2.72	0.589	48
s15850_1-sc4.rep	10386	153	684	273	12031	8414	3543	1.257	0.742	4
s15850_1-sc64.rep	10386	10	684	81	11745	6471	5243	1.905	0.571	64
s15850_1-sc8.rep	10386	77	684	218	11879	8032	3776	1.644	0.709	8

s38417-sc100.rep	23921	17	1742	215	31214	22489	8295	28.232	0.725	100
s38417-sc10.rep	23921	167	1742	759	31514	25922	5261	13.935	0.836	10
s38417-sc16.rep	23921	104	1742	669	31388	25661	5438	19.226	0.827	16
s38417-sc20.rep	23921	84	1742	564	31348	24673	5642	29.188	0.796	20

s38417-sc22.rep	23921	76	1742	662	31332	25454	5528	13.046	0.821	22
s38417-sc24.rep	23921	70	1742	660	31320	25354	5571	13.262	0.817	24
s38417-sc26.rep	23921	64	1742	498	31308	24568	5884	17.164	0.792	26
s38417-sc28.rep	23921	60	1742	559	31300	24412	6078	17.298	0.787	28
s38417-sc2.rep	23921	832	1742	822	32844	27370	5165	15.096	0.882	2
s38417-sc30.rep	23921	56	1742	600	31292	25183	5817	11.645	0.812	30
s38417-sc32.rep	23921	52	1742	475	31284	24264	5992	18.978	0.782	32
s38417-sc34.rep	23921	49	1742	571	31278	25000	5874	13.577	0.806	34
s38417-sc36.rep	23921	47	1742	535	31274	24505	5890	18.737	0.790	36
s38417-sc38.rep	23921	44	1742	431	31268	24181	6120	21.797	0.780	38
s38417-sc3.rep	23921	555	1742	800	32290	26818	5160	13.463	0.865	3
s38417-sc40.rep	23921	42	1742	479	31264	24563	6372	13.707	0.792	40
s38417-sc48.rep	23921	35	1742	382	31250	23871	6492	26.495	0.770	48
s38417-sc4.rep	23921	416	1742	803	32012	26498	5234	14.106	0.854	4
s38417-sc64.rep	23921	26	1742	400	31232	24016	6781	18.483	0.774	64
s38417-sc8.rep	23921	208	1742	766	31596	25984	5311	26.466	0.838	8

s38584_1-sc100.rep	20869	15	1730	148	36333	25140	11184	16.565	0.722	100
s38584_1-sc10.rep	20869	147	1730	524	36597	29763	6816	9.778	0.855	10
s38584_1-sc16.rep	20869	92	1730	412	36487	28819	7639	7.7	0.828	16
s38584_1-sc20.rep	20869	74	1730	457	36451	28950	7488	8.487	0.832	20
s38584_1-sc22.rep	20869	67	1730	455	36437	29023	7395	6.347	0.834	22
s38584_1-sc24.rep	20869	61	1730	360	36425	28320	8062	7.212	0.814	24
s38584_1-sc26.rep	20869	57	1730	322	36417	27993	8348	9.142	0.804	26
s38584_1-sc28.rep	20869	53	1730	366	36409	28485	7907	6.935	0.819	28

s38584_1-sc2.rep	20869	732	1730	530	37767	31109	6616	6.023	0.894	2
s38584_1-sc30.rep	20869	49	1730	410	36401	28416	7971	7.348	0.817	30
s38584_1-sc32.rep	20869	46	1730	351	36395	27921	8447	7.416	0.802	32
s38584_1-sc34.rep	20869	44	1730	384	36391	28428	7942	7.511	0.817	34
s38584_1-sc36.rep	20869	41	1730	277	36385	27386	8952	7.605	0.787	36
s38584_1-sc38.rep	20869	39	1730	372	36381	28171	8187	7.985	0.810	38
s38584_1-sc3.rep	20869	488	1730	547	37279	30546	6709	6.462	0.878	3
s38584_1-sc40.rep	20869	37	1730	379	36377	28256	8109	8.654	0.812	40
s38584_1-sc48.rep	20869	31	1730	278	36365	27445	8890	7.25	0.789	48
s38584_1-sc4.rep	20869	366	1730	530	37035	30218	6777	5.837	0.868	4
s38584_1-sc64.rep	20869	23	1730	205	36349	26217	10074	9.075	0.753	64
s38584_1-sc8.rep	20869	183	1730	512	36669	29770	6880	5.656	0.856	8

s5378-sc100.rep	3050	3	213	8	4557	2251	2306	0.277	0.499	100
s5378-sc10.rep	3050	22	213	84	4595	3655	932	0.123	0.810	10
s5378-sc16.rep	3050	14	213	68	4579	3331	1218	0.276	0.738	16
s5378-sc20.rep	3050	11	213	68	4573	3363	1208	0.143	0.746	20
s5378-sc22.rep	3050	10	213	45	4571	3253	1317	0.327	0.721	22
s5378-sc24.rep	3050	9	213	59	4569	3461	1108	0.364	0.767	24
s5378-sc26.rep	3050	9	213	45	4569	3208	1361	0.272	0.711	26
s5378-sc28.rep	3050	8	213	39	4567	2950	1617	0.42	0.654	28
s5378-sc2.rep	3050	107	213	111	4765	4224	529	0.089	0.936	2
s5378-sc30.rep	3050	8	213	35	4567	3046	1520	0.422	0.675	30
s5378-sc32.rep	3050	7	213	27	4565	2817	1743	0.305	0.624	32
s5378-sc34.rep	3050	7	213	33	4565	2974	1590	0.504	0.659	34

s5378-sc36.rep	3050	6	213	33	4563	3135	1427	0.208	0.695	36
s5378-sc38.rep	3050	6	213	36	4563	3147	1415	0.276	0.698	38
s5378-sc3.rep	3050	72	213	119	4693	4116	562	0.098	0.912	3
s5378-sc40.rep	3050	6	213	32	4563	3025	1537	0.338	0.671	40
s5378-sc48.rep	3050	5	213	21	4561	2952	1609	0.184	0.654	48
s5378-sc4.rep	3050	54	213	102	4659	4042	603	0.103	0.896	4
s5378-sc64.rep	3050	4	213	14	4559	2710	1849	0.24	0.601	64
s5378-sc8.rep	3050	27	213	102	4605	3906	687	0.115	0.866	8

s9234_1-sc100.rep	5844	3	250	8	6933	2264	4669	0.76	0.350	100
s9234_1-sc10.rep	5844	25	250	122	6977	3741	3190	0.643	0.578	10
s9234_1-sc16.rep	5844	16	250	111	6959	3653	3290	0.637	0.564	16
s9234_1-sc20.rep	5844	13	250	93	6953	3372	3564	0.651	0.521	20
s9234_1-sc22.rep	5844	12	250	95	6951	3524	3397	1.046	0.544	22
s9234_1-sc24.rep	5844	11	250	85	6949	3384	3556	0.994	0.523	24
s9234_1-sc26.rep	5844	10	250	84	6947	3246	3692	0.709	0.501	26
s9234_1-sc28.rep	5844	9	250	63	6945	3014	3919	0.921	0.465	28
s9234_1-sc2.rep	5844	124	250	230	7173	4660	2396	0.626	0.720	2
s9234_1-sc30.rep	5844	9	250	73	6945	3120	3818	0.894	0.482	30
s9234_1-sc32.rep	5844	8	250	51	6943	2983	3958	0.689	0.461	32
s9234_1-sc34.rep	5844	8	250	52	6943	2893	4043	0.854	0.447	34
s9234_1-sc36.rep	5844	7	250	46	6941	2970	3966	0.8	0.459	36
s9234_1-sc38.rep	5844	7	250	45	6941	2905	4032	0.732	0.449	38
s9234_1-sc3.rep	5844	83	250	249	7091	4651	2311	0.638	0.718	3
s9234_1-sc40.rep	5844	7	250	40	6941	2844	4097	0.805	0.439	40

s9234_1-sc48.rep	5844	6	250	31	6939	2664	4275	0.747	0.411	48
s9234_1-sc4.rep	5844	62	250	193	7051	4346	2637	0.561	0.671	4
s9234_1-sc64.rep	5844	4	250	16	6935	2584	4351	0.718	0.399	64
s9234_1-sc8.rep	5844	31	250	192	6989	4324	2593	0.606	0.668	8

Table B.2

B.3 Illinois scan - randomized scan-chains

In this section are measured data from ATPG tested on the Illinois-Scan architecture with different number of scan-chains. Scan-chains are generated consistent size and number of scan-chains is for every case 10.

file	gates	iv	ov	patterns	faults	d_faults	r_faults	time	FC	Order
c3540-sc10_10.rep	1719	5	22	10	3438	152	3285	0.339	0.046	1
c3540-sc10_1.rep	1719	5	22	10	3438	152	3285	0.347	0.046	2
c3540-sc10_2.rep	1719	5	22	9	3438	152	3285	0.343	0.046	3
c3540-sc10_3.rep	1719	5	22	9	3438	152	3285	0.348	0.046	4
c3540-sc10_4.rep	1719	5	22	9	3438	152	3285	0.364	0.046	5
c3540-sc10_5.rep	1719	5	22	9	3438	152	3285	0.408	0.046	6
c3540-sc10_6.rep	1719	5	22	9	3438	152	3285	0.372	0.046	7
c3540-sc10_7.rep	1719	5	22	9	3438	152	3285	0.345	0.046	8
c3540-sc10_8.rep	1719	5	22	12	3438	152	3285	0.365	0.046	9
c3540-sc10_9.rep	1719	5	22	12	3438	152	3285	0.343	0.046	10

c5315-sc10_10.rep	2485	18	123	59	5386	1997	3264	0.504	0.377	1
c5315-sc10_1.rep	2485	18	123	43	5386	1814	3480	0.444	0.343	2
c5315-sc10_2.rep	2485	18	123	43	5386	1814	3480	0.51	0.343	3
c5315-sc10_3.rep	2485	18	123	49	5386	1814	3480	0.63	0.343	4
c5315-sc10_4.rep	2485	18	123	41	5386	1814	3480	0.472	0.343	5
c5315-sc10_5.rep	2485	18	123	41	5386	1814	3480	0.613	0.343	6
c5315-sc10_6.rep	2485	18	123	55	5386	1997	3264	0.589	0.377	7
c5315-sc10_7.rep	2485	18	123	55	5386	1997	3264	0.495	0.377	8
c5315-sc10_8.rep	2485	18	123	55	5386	1997	3264	0.511	0.377	9
c5315-sc10_9.rep	2485	18	123	55	5386	1997	3264	0.564	0.377	10

c6288-sc10_10.rep	2448	4	32	15	7752	7215	419	0.288	0.935	1
c6288-sc10_1.rep	2448	4	32	14	7752	7224	446	0.215	0.936	2
c6288-sc10_2.rep	2448	4	32	13	7752	7224	446	0.203	0.936	3
c6288-sc10_3.rep	2448	4	32	13	7752	7224	446	0.225	0.936	4
c6288-sc10_4.rep	2448	4	32	13	7752	7224	446	0.216	0.936	5
c6288-sc10_5.rep	2448	4	32	13	7752	7224	446	0.221	0.936	6
c6288-sc10_6.rep	2448	4	32	13	7752	7224	446	0.26	0.936	7
c6288-sc10_7.rep	2448	4	32	15	7752	7224	446	0.24	0.936	8
c6288-sc10_8.rep	2448	4	32	15	7752	7224	446	0.211	0.936	9
c6288-sc10_9.rep	2448	4	32	15	7752	7215	419	0.279	0.935	10

c7552-sc10_10.rep	3719	21	108	16	7592	728	6745	1.402	0.098	1
c7552-sc10_1.rep	3719	21	108	17	7592	708	6768	1.62	0.095	2
c7552-sc10_2.rep	3719	21	108	23	7592	708	6768	1.275	0.095	3
c7552-sc10_3.rep	3719	21	108	19	7592	708	6768	1.318	0.095	4
c7552-sc10_4.rep	3719	21	108	18	7592	708	6768	1.448	0.095	5
c7552-sc10_5.rep	3719	21	108	22	7592	708	6768	1.341	0.095	6
c7552-sc10_6.rep	3719	21	108	25	7592	708	6768	1.268	0.095	7
c7552-sc10_7.rep	3719	21	108	23	7592	708	6768	1.533	0.095	8
c7552-sc10_8.rep	3719	21	108	24	7592	708	6768	1.317	0.095	9
c7552-sc10_9.rep	3719	21	108	20	7592	708	6768	1.244	0.095	10

s13207_1-sc10_10.rep	8679	70	790	188	9955	6255	3685	0.813	0.647	1
s13207_1-sc10_1.rep	8679	70	790	247	9955	6523	3414	0.875	0.675	2

s13207_1-sc10_2.rep	8679	70	790	243	9955	6523	3414	0.906	0.675	3
s13207_1-sc10_3.rep	8679	70	790	261	9955	6620	3308	1.184	0.685	4
s13207_1-sc10_4.rep	8679	70	790	278	9955	6620	3308	0.953	0.685	5
s13207_1-sc10_5.rep	8679	70	790	264	9955	6620	3308	0.909	0.685	6
s13207_1-sc10_6.rep	8679	70	790	272	9955	6620	3308	0.969	0.685	7
s13207_1-sc10_7.rep	8679	70	790	178	9955	6255	3685	0.824	0.647	8
s13207_1-sc10_8.rep	8679	70	790	178	9955	6255	3685	0.972	0.647	9
s13207_1-sc10_9.rep	8679	70	790	178	9955	6255	3685	0.918	0.647	10

s15850_1-sc10_10.rep	10386	62	684	231	11847	8016	3760	1.583	0.707	1
s15850_1-sc10_1.rep	10386	62	684	276	11847	8144	3636	1.705	0.718	2
s15850_1-sc10_2.rep	10386	62	684	250	11847	8047	3686	1.728	0.710	3
s15850_1-sc10_3.rep	10386	62	684	258	11847	8047	3686	1.7	0.710	4
s15850_1-sc10_4.rep	10386	62	684	254	11847	8046	3686	1.802	0.710	5
s15850_1-sc10_5.rep	10386	62	684	236	11847	8020	3761	1.803	0.707	6
s15850_1-sc10_6.rep	10386	62	684	239	11847	8020	3761	1.617	0.707	7
s15850_1-sc10_7.rep	10386	62	684	235	11847	8021	3761	1.337	0.708	8
s15850_1-sc10_8.rep	10386	62	684	225	11847	8021	3761	1.394	0.708	9
s15850_1-sc10_9.rep	10386	62	684	226	11847	8015	3760	1.284	0.707	10

s38417-sc10_10.rep	23921	167	1742	746	31514	25938	5281	11.076	0.836	1
s38417-sc10_1.rep	23921	167	1742	720	31514	25889	5301	10.679	0.835	2
s38417-sc10_2.rep	23921	167	1742	766	31514	25843	5392	11.257	0.833	3
s38417-sc10_3.rep	23921	167	1742	744	31514	25870	5330	10.754	0.834	4
s38417-sc10_4.rep	23921	167	1742	767	31514	25867	5330	10.542	0.834	5

s38417-sc10_5.rep	23921	167	1742	616	31514	24798	5721	15.165	0.800	6
s38417-sc10_6.rep	23921	167	1742	763	31514	25860	5348	10.464	0.834	7
s38417-sc10_7.rep	23921	167	1742	754	31514	25866	5348	10.829	0.834	8
s38417-sc10_8.rep	23921	167	1742	764	31514	25935	5273	10.079	0.836	9
s38417-sc10_9.rep	23921	167	1742	764	31514	25932	5273	10.382	0.836	10

s38584_1-sc10_10.rep	20869	147	1730	466	36597	29426	7146	6.223	0.846	1
s38584_1-sc10_1.rep	20869	147	1730	530	36597	29765	6809	6.143	0.855	2
s38584_1-sc10_2.rep	20869	147	1730	541	36597	29765	6809	5.989	0.855	3
s38584_1-sc10_3.rep	20869	147	1730	523	36597	29778	6803	5.731	0.856	4
s38584_1-sc10_4.rep	20869	147	1730	452	36597	29444	7117	5.826	0.846	5
s38584_1-sc10_5.rep	20869	147	1730	451	36597	29444	7117	5.839	0.846	6
s38584_1-sc10_6.rep	20869	147	1730	461	36597	29384	7192	5.619	0.844	7
s38584_1-sc10_7.rep	20869	147	1730	497	36597	29419	7164	5.837	0.845	8
s38584_1-sc10_8.rep	20869	147	1730	489	36597	29419	7164	5.997	0.845	9
s38584_1-sc10_9.rep	20869	147	1730	464	36597	29426	7146	5.774	0.846	10

s5378-sc10_10.rep	3050	22	213	96	4595	3864	729	0.112	0.857	1
s5378-sc10_1.rep	3050	22	213	94	4595	3771	816	0.139	0.836	2
s5378-sc10_2.rep	3050	22	213	94	4595	3771	816	0.136	0.836	3
s5378-sc10_3.rep	3050	22	213	94	4595	3771	816	0.136	0.836	4
s5378-sc10_4.rep	3050	22	213	94	4595	3771	816	0.137	0.836	5
s5378-sc10_5.rep	3050	22	213	94	4595	3771	816	0.138	0.836	6
s5378-sc10_6.rep	3050	22	213	94	4595	3771	816	0.136	0.836	7
s5378-sc10_7.rep	3050	22	213	97	4595	3864	729	0.114	0.857	8

s5378-sc10_8.rep	3050	22	213	97	4595	3864	729	0.126	0.857	9
s5378-sc10_9.rep	3050	22	213	97	4595	3864	729	0.115	0.857	10

s9234_1-sc10_10.rep	5844	25	250	175	6977	3942	2978	0.71	0.609	1
s9234_1-sc10_1.rep	5844	25	250	140	6977	3781	3137	0.661	0.584	2
s9234_1-sc10_2.rep	5844	25	250	140	6977	3781	3137	0.662	0.584	3
s9234_1-sc10_3.rep	5844	25	250	134	6977	3781	3137	0.66	0.584	4
s9234_1-sc10_4.rep	5844	25	250	134	6977	3781	3137	0.658	0.584	5
s9234_1-sc10_5.rep	5844	25	250	173	6977	3942	2978	0.694	0.609	6
s9234_1-sc10_6.rep	5844	25	250	173	6977	3942	2978	0.691	0.609	7
s9234_1-sc10_7.rep	5844	25	250	178	6977	3942	2978	0.684	0.609	8
s9234_1-sc10_8.rep	5844	25	250	183	6977	3942	2978	0.715	0.609	9
s9234_1-sc10_9.rep	5844	25	250	183	6977	3942	2978	0.712	0.609	10

Table B.3

Appendix C

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources, samples - generated Illinois-Scan, configuration of scan-chains, structures
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format