Circuits and Systems, 2013, 4, 342-349
http://dx.doi.org/10.4236/cs.2013.44046 Published Online August 2013 (http://www.scirp.org/journal/cs)

Study on Test Compaction in High-Level Automatic Test Pattern Generation (ATPG) Platform

Ayub Chin Abdullah, Chia Yee Ooi
Malaysia-Japan International Institute of Technology, Universiti Teknologi Malaysia, Kuala Lumpur, Malaysia
Email: yob_kejor@yahoo.com, ooichiayee@ic.utm.my

Received January 14, 2013; revised February 4, 2013; accepted February 11, 2013

ABSTRACT

Advancements in semiconductor technology are making gate-level test generation more challenging. This is because a large amount of detailed structural information must be processed in the search process of automatic test pattern generation (ATPG). In addition, ATPG needs to deal with new defects caused by process variation when IC is shrinking. To reduce the computation effort of ATPG, test generation could be started earlier at higher abstraction level, which is in line with top-down design methodology that has become more popular nowadays. In this research, we employ Chen's high-level fault model in the high-level ATPG. Besides shorter ATPG time as shown in many previous works, our study showed that high-level ATPG also contributes to test compaction. This is because most of the high-level faults correlate with the gate-level collapsed faults especially at input/output of the modules in a circuit. The high-level ATPG prototype used in our work is mainly composed by constraint-driven test generation engine and fault simulation engine. Experimental result showed that more reduced/compact test set can be generated from the high-level ATPG.

Keywords: Automatic Test Pattern Generation (ATPG); Constraint Logic Programming (CLP); Verilator; Circuit-Under-Test (CUT); Test Compaction

1. Introduction

The role of testing in integrated circuit (IC) is to determine the correctness of manufactured circuits. Therefore, testing is important since the fraction of good chips sold in the market yields the quality of the product. In very large scale integration (VLSI) realization process, test development is done after the IC design and verification process, as can be seen in **Figure 1(a)**. According to Moore's Law, the number of transistors in IC doubles every 18 months [1]. Nowadays, an IC easily can have millions of gates [2]. In order to cope with this situation, the top-down design methodology starting with high-level description had been introduced.

The test generation algorithms designed for the high-level models are usually direct extensions of those for the gate-level models, in which the functional modules are treated as primitive components so fewer components are evaluated during test generation. Since digital design is first described at high-level, the test generation could be done earlier even before the design is synthesized into gate-level circuit. The resulting reduced structural complexity makes high-level test generation attractive. Thus, functional test generation and functional fault simulation of a digital system described in high-level models offers an attractive alternative in test development.

2. Previous Works

It has been known that real-world problems could be defined mathematically and logically. Since Constraint Logic Programming (CLP) is based on logical reasoning, it has been used to solve real-world problem. It started with solving the puzzle problem. After that, it moves through main business, management and industrial combinatorial applications such as resource allocation, timetabling, crew rostering, scheduling, planning, vehicle routing and others. CLP application could be used in ATPG since ATPG is a search problem for the test patterns of digital circuit. This is being done by defining the ATPG problem mathematically, and the solution of the defined ATPG problem is the test patterns. The work in [3] uses CLP to find test patterns for the faults undetected by random engine and transition-oriented engine. It could be seen clearly in **Figure 1(b)**.

The random engine together with the simulation-based

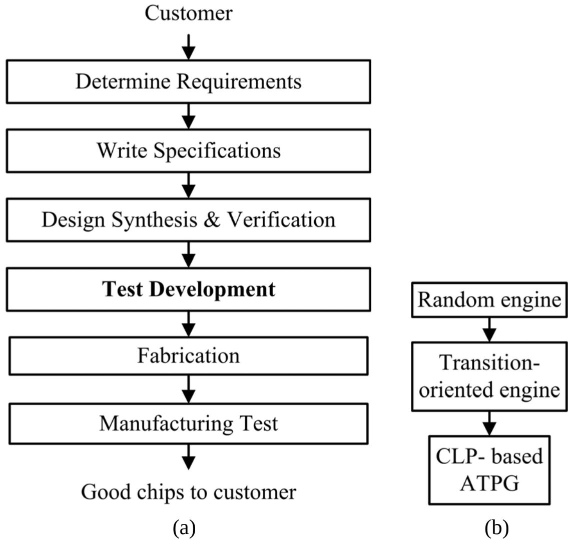


Figure 1. Testing flow (a) in VLSI realization process (b) ATPG in [3].

ATPG was used to find the test sequence. The transition-oriented engine [4] finds input stimuli that can cause the transition of fault-free circuit differ from the faulty circuit even though its effect towards PO is not proved yet. Transition refers to the movement of the state Register from current state to the next state. A high-level fault model was described in signals and condition statement.

An explanation of this kind of fault model was not included but it looked like the *input stuck-at-fault* and *if stuck-at-fault* in work [2]. The ATPG in [4] has been tested on the ITC‘99 benchmark where the high-level fault coverage and test length were presented. However, the test patterns generated from the ATPG has not been tested on gate-level circuits for the gate-level fault coverage. So, the effectiveness of the test pattern cannot be analyzed.

The work in [5] proposed an ATPG approach at behavioral model using CLP. The VHDL circuit is first converted to decision diagram (DD) model. DD model is another way to represent a digital circuit. A high-level fault model was defined for the faults at branch and operator of the CUT. The faults were injected to the DD by creating several faulty DDs. The fault model is equivalent to “if stuck-at-fault” and “micro-operation fault” in [2]. The method was tested on GCD, FFT, SQRT and DIFFEQ circuits. Although the high-level fault coverage achieved was greater than 90% for all circuits but the number of injected faults was low which is mostly below 30 except the faults injected to DIFFEQ circuit which is 66 faults. The test length was also not presented. Besides, comparison of this ATPG with the gate-level ATPG and other methods of high-level ATPG were not presented.

The work in [6] proposed a CLP-based ATPG using 3-valued logic, 9-valued logic and 11-valued logic. The 3-valued logic refers to the traditional line assignment which is (0, 1, *x*) where “*x*” is the “don’t care” value. “*x*” could be 0 or 1. 9-valued logic refers to MUTH algebra logic assignment which includes 0, 1, *D*, *D*, *x*, 0/*x*, 1/*x*, *x*/0 and *x*/1. 11-valued logic is the extension of MUTH algebra where two additional logic assignments are added, namely “*e*” and “*e*”. “*e*” and “*e*” refers to “*D*” and “*D*” that occurs at re-convergent fan-out. These lines usually have many possibilities of value during fault propagation. The method was tested on ISCAS‘85 benchmark where 3-valued logic, 9-valued logic and 11-valued logic were used as the propagation rule. The gate-level fault coverage was 100% for circuits which consist of about 10 gates and only 28% for circuit with 400 gates. The test length for circuits below 10 gates only was presented. Besides, no comparison with other methods was presented.

Test compaction refers to the process of reducing the number of test patterns in a test set without reducing its fault coverage [7]. Several approaches have been done in compacting test patterns. The approaches include the static test compaction, dynamic test compaction, and fault collapsing. In static test compaction, the test patterns are compacted after the gate-level test pattern generation process and it is also called post-generation compaction technique. In dynamic test compaction, the compaction technique is done along with the gate-level test generation process. Fault collapsing which is the process of reducing the size of fault list with two concepts which are fault equivalence and fault dominance, is also one of the techniques in compacting test patterns [8]. Two faults are called equivalent if every pattern that detects one of the faults also detects the other. A fault, f1 dominates another fault, f2 if the test set for f2, T2 is a subset of the test set for f1, T1. Equivalence fault collapsing and dominance fault collapsing can be used to aid test compaction [1].

The work in [9] proposed a mixed-mode static compaction technique. It includes the restoration-based and omission-based compaction technique. In restoration-based technique, a subset of the previous saved test patterns used or restored on the next undetected fault so the number of test patterns could be decreased during ATPG. In the omission-based compaction technique, certain test vectors are removed from test sequences in order to achieve smaller test length. The test vectors are omitted one at a time from the beginning of the test sequence and fault simulation is performed. If the same faults continue to be detected after omitting a vector, the vector is removed from the sequence. Otherwise, the vector is retained. The method has been tested on ISCAS‘89 and ADDENDUM‘93 benchmarks. The test patterns are from the STRATEGATE test generator [10]. Although, a good compaction ratio is achieved in term of test length, no fault coverage has been presented.

3. Functional Fault Model

Functional faults are defined at functional level. A functional fault changes the function of a component or functional block. In this research, Chen's functional fault model [2] which consists of 10 types of faults has been used. The definition for each fault is as follow.

1) **Input stuck-at fault**: The input signal is stuck-at-0 or 1. The input stuck-at fault represents the failure of the primary input signal. The input signal can be stuck-at-0 or 1. This fault is mapped by replacing every occurrence of the input signal in the Verilog file with a corresponding stuck-at-fault.

2) **Output stuck-at fault**: The output stuck-at-fault represents the failure of the primary output signal. The output signal can be stuck-at-0 or 1. This fault is mapped by replacing the right hand side of all occurrences of the output signal assignment in the Verilog file with the corresponding stuck-at-fault.

3) **If stuck then fault**: The if stuck then fault represents the failure to execute the "else" and "else if" portion of statements for the "if" construct. This fault is mapped by replacing the condition in "if" with the value "1".

4) **If stuck else fault**: The if stuck else fault represents the failure to execute the "if" portion of statements for the "if" construct. This fault is mapped by replacing the condition in "if" with value 0.

5) **Else if stuck then fault**: The else if stuck then fault represents the failure to execute the following "else" and "else if", if it exist portion of statements for the "if" construct. This fault is mapped by replacing the condition in "else if" with value "1".

6) **Else if stuck else fault**: The else if stuck else fault represents the failure to execute the "else if" portion of statements for the "if" construct. This fault is mapped by replacing the condition between "else if" with value "0".

7) **Assignment statement fault**: The assignment statement fault represents the failure to assign a new value to a signal. In the presence of an assignment statement fault, the signal to the left side of the assignment operator (<=) will be assigned to one of the logic values the signal can have. The fault is mapped by replacing the expression to the right of the assignment operator with a corresponding logic value, for example "0" and "1" for the 1 bit type.

8) **Dead clause fault**: The dead clause fault represents a failure of a value in a switch-case statement to execute when selected. The mapping of the fault is done by replacing the expression to the right of the assignment operator with the signal name to the left of the operator in the corresponding clause. However, if the signal to the left of the operator is an output, then the fault is mapped by commenting the assignment statement in that clause.

9) **Micro-operation fault**: The micro-operation fault represents a failure of a micro-operation to perform its intended function. The operator can be classified into four categories: logical operators, relational operators, unary operators, and arithmetic operators. An operator may fail to any other operator in its category. This fault is mapped by replacing the operator considered with its counter operator that must be defined.

10) **Local stuck data fault**: The local stuck data fault represents a failure for a signal object to have a proper value within a local expression. More than one expression within a device model may use the signal. The fault is mapped such that the signal in only one expression will be replaced with one of the logic values that the signal can have. The signal in other expressions will retain the proper logic value.

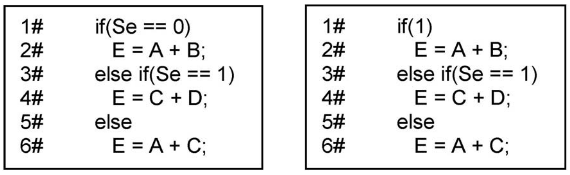
4. Test Compaction Using Functional Fault Model

This section explains how test compaction occurs in a high-level ATPG using Chen's functional fault model. Chen's fault model defines several types of faults as elaborated in Section 3, most of which are injected at the input/output of the modules in a circuit described at functional level. These functional faults are mapped to the gate-level stuck-at faults at inputs of a module. Besides that, we also analyze that micro-operation faults correlate with some stuck-at faults. The correlation contributes to gate-level test compaction, which can be proved by checkpoint theorem theoretically. Checkpoints are defined as primary inputs and fan-out branches of a circuit which have been proposed as starting set of faults for both equivalence and dominance fault collapsing [1].

Checkpoint Theorem: A test set which detects all single stuck-at faults of the checkpoints of a combinational circuit detects all single stuck-at faults in that circuit.

The following text relates functional faults to the gate-level collapsed faults at checkpoints. Primary inputs are checkpoints in a circuit-under-test at gate-level, whose faults are dominated by some faults inside the circuit. In other words, test patterns of faults at primary inputs can detect other faults inside the circuits. When we view the CUT at high-level, composed by several modules, test patterns of input stuck-at-fault in a functional fault model are compacted test patterns. On elementary gates such as AND gate, the fault on its output is equivalent on to the faults on its inputs. Those inputs could be fan-out branches inside a circuit viewed on gate-level if the AND gate's output is also the primary output. Therefore, the fault collapsing could be done by injecting fault at the primary outputs using Chen's output stuck-at fault.

The following explains test compaction via *if stuck then fault*. **Figure 2(a)** shows statements of "if", "else if" and "else" written in Verilog language for fault-free circuit while **Figure 2(b)** shows the statements for faulty circuit. The schematic diagram of the Verilog statements of **Figure 2(a)** could be seen in **Figures 3(a)** and **4(a)**. While the schematic diagram of the Verilog statements of **Figure 2(b)** could be seen **Figures 3(b)** and **4(b)**. **Figure 3** show the schematic diagrams at high-level while **Figure 4** show the schematic diagrams at gate-level. When *If stuck then fault* is injected to the Verilog statement in **Figure 2(a)** is represented in **Figure 2(b)**. This fault is illustrated at high-level in **Figure 3(b)**, which is also equivalent to the gate-level s-a-0 fault in **Figure 4(b)**. This example deduced that the *if stuck-then-fault* is mapped to the gate-level fault at checkpoint Se[0], which allow test compaction.



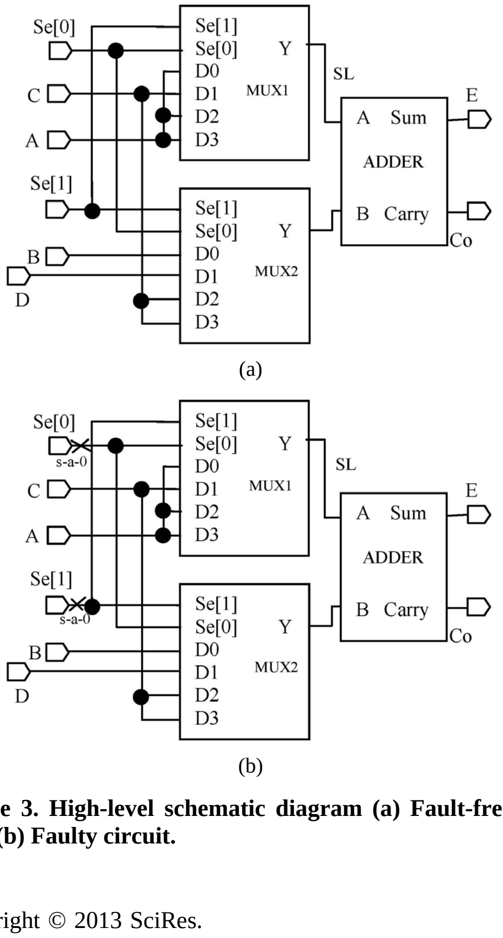
```
1#      if(Se == 0)
2#          E = A + B;
3#      else if(Se == 1)
4#          E = C + D;
5#      else
6#          E = A + C;
```

(a)

```
1#      if(1)
2#          E = A + B;
3#      else if(Se == 1)
4#          E = C + D;
5#      else
6#          E = A + C;
```

(b)

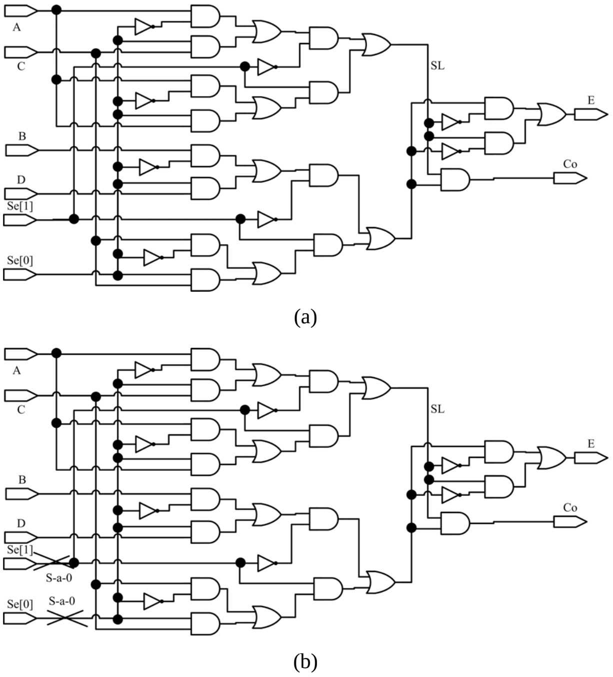
Figure 2. An example of if-else statements in Verilog (a) Fault-free circuit (b) Faulty circuit.



(a)

(b)

Figure 3. High-level schematic diagram (a) Fault-free circuit; (b) Faulty circuit.



(a)

(b)

Figure 4. Gate-level schematic diagram (a) Fault-free circuit (b) Faulty circuit.

Similar analysis and discussion can be used for *if stuck else fault*, *else if stuck then fault*, and *else if stuck else fault* from Chen' fault model. For *if stuck else fault*, the Verilog statements inside the bracket of the first line from **Figure 2(a)** is replaced with 0 for fault injection, so the primary input of Se is stuck-at-1. In else if stuck then fault, the bracket inside the third line of **Figure 2(a)** is replaced with 1, so Se stuck-at-1 while the *else if stuck else fault* replaces the third line of **Figure 2(a)** with 0. Hence, Se stuck-at-2 or Se[1] stuck-at-1 and Se[0] stuck-at-0.

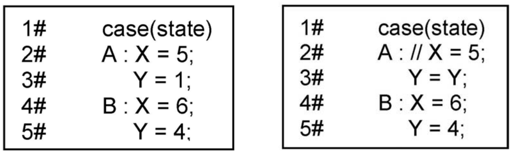
The test compaction for the *assignment statement fault* of E = 0 + B at line 2 in **Figure 2(a)** is done by injecting a stuck-at-0 at the output module of Mux1 called SL in both **Figures 3(a)** and **4(a)**. By using the fault equivalence rule for the gate-level circuit in **Figure 3(a)**, the injected line of the *assignment statement fault* which is an output of a gate carries a stuck-at fault that could be equivalent to input of gates inside the Mux module. Those gate input lines could be the fan-out branches in the whole circuit where according to checkpoint theorem, it could be collapsed.

The following explains the test compaction for the *dead clause fault*. **Figure 5(a)** shows portions of "switch-case" statements written in Verilog while **Figure 5(b)** shows the faulty Verilog statements with dead clause fault which is stuck-at state A. The statement in line 2 is commented and the statement in line 3 is overwritten such that signal Y holds the same value.

The schematic diagram of both **Figures 5(a)** and **(b)** can be seen in **Figure 6**. The *dead clause fault* can be equivalent to gate-level s-a-0 at line LD which is the enable signal to load the D flip-flop. The test compaction for the *local stuck data fault* is done by injecting a stuck-at-0 or stuck-at-1 at the input module of Adder called SL in both **Figures 3(a)** and **4(a)**. The location of this injected line is at the checkpoint of the whole Adder module which could aid in test compaction.

5. High-Level ATPG Platform

In this research, Constraint Logic Programming (CLP)-based high-level ATPG platform is used to demonstrate test compaction using functional fault model. Verilator is an open source which is used in our CLP-based ATPG as it Verilator-based fault simulator [11]. **Figure 7** shows the platform of the proposed high-level ATPG in this research while **Figure 8** shows how this ATPG operates. This ATPG could be divided into three main blocks that represent fault injection, ECLiPSe-based ATPG and Verilator-based fault simulation respectively as in **Figure 7**. The fault injection process is to create fault list and the faulty circuits in Verilog from the circuit-under-test (CUT). These faulty circuits and CUT, will be passed to the ECLiPSe-based ATPG and Verilator-based fault simulation blocks. The ECLiPSe-based ATPG is to search the test patterns for the corresponding faulty circuit and compute its test application time or test length. The generated test patterns from the ECLiPSe-based ATPG will then be passed to the Verilator-based fault simulator. The Verilator-based fault simulator will compute the fault coverage based on the given test patterns. Besides that, the fault list will be updated by dropping the detected faults and the test patterns will be saved correspondingly. If the fault list is empty, the ATPG will be terminated and if it is not, the process from the ECLiPSe-based ATPG will be repeated.



```
1#     case(state)
2#     A : X = 5;
3#         Y = 1;
4#     B : X = 6;
5#         Y = 4;
```

(a)

```
1#     case(state)
2#     A : // X = 5;
3#         Y = Y;
4#     B : X = 6;
5#         Y = 4;
```

(b)

Figure 5. An example of switch-case statements written in Verilog (a) Fault-free circuit (b) Faulty circuit

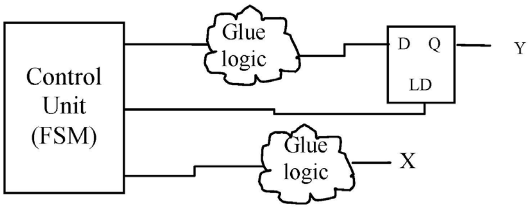


Figure 6. Schematic diagrams of “switch-case” statements.

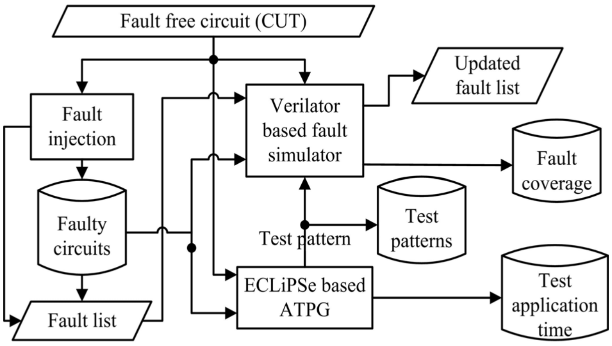


Figure 7. ATPG platform.

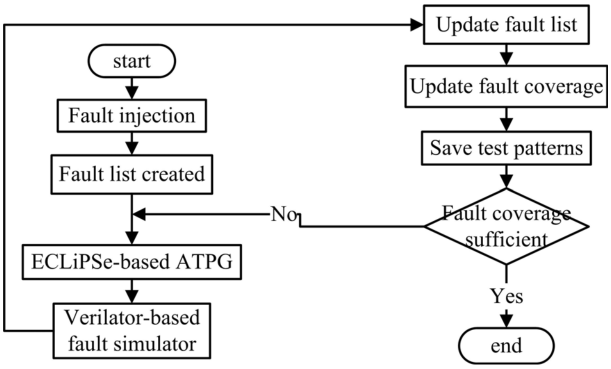


Figure 8. ATPG operation.

6. Experiment Result and Result Discussion

The experiment began by converting the VHSIC Hardware Description Language (VHDL) file of the International Test Conference (ITC)‘99 benchmark circuits [9] to Verilog files. ITC‘99 benchmark circuits are described at functional level. Vhd2vl [12] was used to convert VHDL file to Verilog file because Verilator is a tool that can converts only Verilog to C++ file. Eight circuits have been tested, which are b01, b02, b03, b04, b06, b09, b10 and b11.

Five experiments have been conducted in this research. The first experiment is invoking the functional Automatic Test Pattern Generation (ATPG) to generate the functional test patterns for the circuit-under-tests (CUTs). The test patterns were saved in Verilog testbench format while fault coverage and test application time were saved in text file format. The effectiveness of functional test patterns as compacted test patterns will be shown in Experiments 3-5.

After that, the gate-level netlists of the benchmark circuits were obtained using synthesis tool called Synopsys Design Vision. The second experiment began with Synopsys ATPG called TetraMax. The test patterns, fault coverage and test application time for the gate-level netlists were then saved. The purpose of this experiment is to obtain the fault coverage of the benchmark circuits.

The fault coverage in this experiment is maximal since it is conducted using the powerful commercial gate-level ATPG and it should be achievable even in the case where test compaction is applied. Another purpose of this experiment is to obtain test length to be compared with the first experiment.

For the third experiment, fault simulation was invoked in TetraMax to simulate the netlists using functional ATPG test patterns to evaluate the quality of the test patterns on detection of gate-level faults. The detected fault list was saved. In the fourth experiment, TetraMax was invoked again to generate test patterns for the faults remain undetected after fault simulation that used functional test patterns. Subsequently, total fault coverage and test application time or test length were known.

For the last experiment, Synopsys ATPG was invoked for the saved detected fault list of Experiment 3. This experiment is to show that the reduced test set (functional ATPG test patterns) can detect as many faults as the gate-level test set that are bigger. The graphical view of the experiment could be seen in **Figure 9**.

6.1. Fault Coverage

Fault coverage is the percentage of detected faults among the total faults. The formula is shown in Equation (1). **Table 1** shows the fault coverage obtained in this research. The third column shows the number of injected functional faults while the fourth column shows the obtained high-level fault coverage.

The second column shows the number of injected gate-level faults using the Synopsys TetraMax. The fourth, fifth and sixth columns show the gate-level fault coverage obtained from Experiments 2-4 respectively. The gate-level fault coverage obtained from both Experiments 2 and 4 are the same for all circuits which

$$\text{Fault coverage} = \frac{\text{Faults detected}}{\text{Number of faults}} \times 100\% \qquad (1)$$

are the maximum fault coverage for the respective benchmarks circuits. The difference between these two experiments is that Experiment 2 used merely gate-level test patterns while Experiment 4 used the mixture of gate-level test patterns and functional test patterns. In other words, this experiment needs combination of gate-level ATPG and functional ATPG. The gate-level fault coverage obtained for Experiment 3 is high except for b01 and b03. b01 and b03 have low fault coverage owing to the numbers of injected functional faults that is much lower than the number of their gate-level faults. This could cause many gate-level faults evaded in Synopsys ATPG because the correlation between the functional faults and the gate-level faults is very weak.

6.2. Test Application Time

Test application time is the number of clock cycles needed for the stored test patterns to test a CUT. **Table 2(a)** shows the test application time obtained in Experiments 1, 2 and 4 respectively. Test compaction is conducted in order to reduce the data need to be sent to and stored in the automatic test equipment (ATE). This can

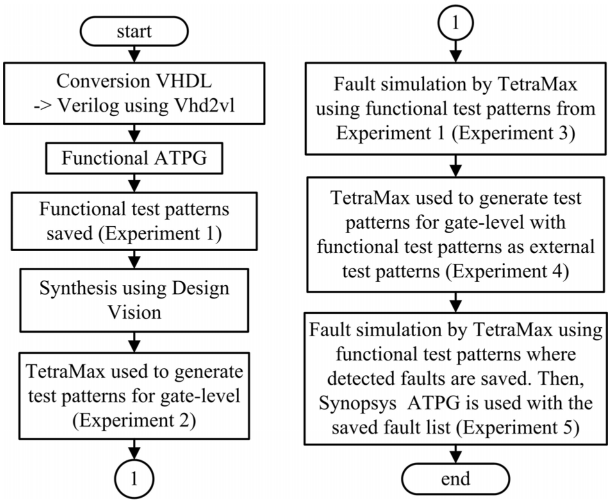

start
Conversion VHDL -> Verilog using Vhd2vl
Functional ATPG
Functional test patterns saved (Experiment 1)
Synthesis using Design Vision
TetraMax used to generate test patterns for gate-level (Experiment 2)
1
1
Fault simulation by TetraMax using functional test patterns from Experiment 1 (Experiment 3)
TetraMax used to generate test patterns for gate-level with functional test patterns as external test patterns (Experiment 4)
Fault simulation by TetraMax using functional test patterns where detected faults are saved. Then, Synopsys ATPG is used with the saved fault list (Experiment 5)
end

Figure 9. Experimental setup.

Table 1. Fault coverage (%).

Circuit	# gates faults	# func. faults	HL FC in Ex. 1 (%)	GL FC in Ex. 2 (%)	GL FC in Ex. 3 (%)	GL FC in Ex. 4 (%)
b01	274	68	100.00	99.27	37.96	99.27
b02	196	24	79.17	89.80	68.88	89.80
b03	1042	46	64.00	69.00	25.38	69.00
b04	3586	70	57.14	89.27	60.10	89.27
b06	338	37	85.71	96.75	90.10	96.75
b09	960	24	58.33	88.18	64.20	88.18
b10	1110	90	60.10	94.32	70.10	94.32
b11	2842	111	56.90	81.76	60.20	81.76

Table 2. Test application time (clock cycles) (a) Gate-level test patterns (b) High-level test patterns.

(a)

Circuit	TAT Ex. 1	TAT Ex. 2	TAT Ex. 4	Compaction ratio (%)
b01	7	68	66	2.94
b02	15	34	33	2.94
b03	6	80	79	1.25
b04	15	118	114	3.39
b06	26	38	33	13.16
b09	35	827	813	1.69
b10	105	251	232	7.57
b11	102	201	180	10.45
			Average	5.42

(b)

Circuit	# Faults	TAT Ex. 3	TAT Ex. 5	Compaction ratio (%)
b01	104	7	9	22.22
b02	135	15	16	6.25
b03	264	6	7	14.29
b04	1791	15	18	16.67
b06	305	26	31	16.13
b09	405	35	39	10.26
b10	778	105	124	15.32
b11	1711	102	123	17.07
			Average	14.78

be seen by calculating the compaction ratio of it as in Equation (2) [8]. Column 5 shows the compaction ratio obtained.

$$\text{Compaction ratio} = \frac{(\text{Original test length} - \text{current test length})}{\text{Original test length}} \times 100\% \quad (2)$$

The greater the compaction ratio of it makes the data sent to the ATE reduced. ATE is the IC tester used during the test manufacturing. The original test length in Equation (2) refers to the test application time (column 3) from the gate-level ATPG (Experiment 2). The current test length refers to the test application time (column 4) from the combination of gate-level ATPG and functional ATPG using fault model in [2] (Experiment 4). Test length is another terms used for test application time. Compaction ratio greater than 10% has been achieved for circuit b06 and b11. This is because the correlation between functional fault and gate-level fault is high.

Experiment 5 is to show that the test compaction was resulted from the functional ATPG. In other words, functional ATPG in our work can generate compact test patterns that cover subset of the faults in the benchmarks circuits. The second column of **Table 2(b)** shows the number of gate-level faults in those CUT that detected in Experiment 3 by the functional test patterns. Its test application time is shown in the third column. Test application time of gate-level test patterns that can detect the same set of faults is shown in the fourth column. The compaction ratio is displayed in the fifth column. The faults involved are subset of the faults detectable by the functional test patterns generated in Experiment 1. From **Table 2(b)**, the average compaction ratio achieved is 16.13%. Only for circuits b01 and b02 that achieved 22.22% and 6.25%.

7. Conclusions

The study has shown that ATPG using functional fault model allows test compaction during test generation process whereby the test patterns can cover subset of the gate-level faults. The remaining undetectable faults can be covered by gate-level ATPG. All in all, the complete test patterns using both functional ATPG and gate-level ATPG are more compacted. Experiments on eight benchmark circuits of ITC‘99 circuits have been conducted. The experiments involved the ECLiPSe-based ATPG system, Synopsys TetraMax gate-level ATPG system, and the combination of both ATPG systems. Test length reduction from the original gate-level ATPG system has been achieved up to 16.13%.

Several potential extensions can be carried out to the work in this research in the interest of achieving better performance. These are suggested below.

1) Computation time—The computation is higher than the gate-level ATPG where an improvement needed to decrease it.

2) Graphical User interface (GUI)—In this proposed hybrid ATPG system, a command-line just used as the user interface. A friendly GUI is necessary, so that users can monitor the test generation process, or generate test vectors for a particular fault easily.

3) Test environment—A test environment that using the CLP and Perl has been created, however the experiments only conducted on the fault coverage and test length. An experiment on test environment coverage proposed for future research.

4) Design-for-testability (DFT)—In this research, a method to generate test patterns proposed. DFT have the capacity to improve the test generation. A future research on this should be conducted.

8. Acknowledgements

The work was supported by Universiti Teknologi Malaysia under Research University Grant (No. Q.K130000. 2643.05J63).

REFERENCES

[1] B. L. Michael and A. D. Vishwani, “Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits,” Kluwer Academic Publishers, New York, 2002.

[2] H. C. Chien-In, “Behavioral Test Generation/Fault Simulation,” *IEEE Potentials*, Vol. 22, No. 1, 2003, pp. 27-32. doi:10.1109/MP.2003.1180938

[3] G. D. Giuseppe, F. Franco, *et al.*, “Test Generation Based on CLP,” 8*th International Workshop on Micro- processor Test and Verification*, *Common Challenges and Solutions*, Austin, 5-6 December 2009, pp. 98-105.

[4] D. G. Guglielmo, F. Fummi, C. Marconcini and G. Pravadelli, “Improving High-Level and Gate-Level Testing with FATE: A Functional Automatic Test Pattern Generator Traversing Unstabilised Extended FSM,” *IET Computers & Digital Techniques*, Vol. 1, No. 3, 2007, pp. 187-196. doi:10.1049/iet-cdt:20060139

[5] Y. Sun, “Automatic Behavioral Test Generation by Using a Constraint Solver,” Master’s Thesis, Linköping University, Linköping, 2001.

[6] S. Brand, “Sequential Automatic Test Pattern Generation by Constraint Programming,” *Proceedings of CP* 2001 *Workshop on Modelling and Problem Formulation*, Cyprus, 1 December 2001, pp. 1-8.

[7] S. D. Hochbaum, “An Optimal Test Compression Procedure for Combinational Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 15, No. 10, 1996, pp. 1294-1299.

[8] N. Zainalabedin, “Digital System Test and Testable Design Using HDL Models and Architectures,” Springer, New York, 2011.

[9] R. Guo, S. M. Reddy, *et al.*, “Reverse-Order-Restoration-Based Static Test Compaction for Synchronous Sequential Circuits,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 22, No. 3, 2003, pp. 293-304.

[10] M. S. Hsiao, E. M. Rudnick and J. H. Patel, “Sequential Circuit Test Generation Using Dynamic State Traversal,” *Proceedings of the* 1997 *European Design and Test Conference*, Paris, 17-20 March 1997, pp. 22-28.

[11] W. Snyder, “Verilator-3.810,” Veripool, 2010.

[12] F. Corno, S. M. Reorda and G., Squillero, “RT-Level ITC‘99 Benchmarks and First ATPG Results,” *IEEE Design & Test of Computers*, Vol. 17, No. 3, 2000, pp. 44-53. doi:10.1109/54.867894.