Approximate Fault Simulation for Quick Evaluation of Test Patterns in Digital Circuit Testing

Leili Khosravi [1], Arezoo Kamran [2*]

Department of Computer Engineering and Information Technology, Razi University, Kermanshah, Iran

[1]leyli.khosravi536@gmail.com, [2]kamran@razi.ac.ir[*]

[*]Corresponding author

Received: 2021-07-28
Accepted: 2022-01-19

Abstract

Simulation-based test pattern generation methods are an interesting alternative to deterministic methods because of lower time complexity. In these methods, test patterns are evaluated and those with higher efficiency are selected. Traditionally, test pattern selection is based on fault coverage, which is an accurate merit indicator, but its calculation is time-consuming. Instead of fault coverage, approximate indicators can be used to assess efficiency of test patterns. In this paper, an approximate indicator called *APXD* is proposed, which is more efficient than existing approximate methods. Our experimental results show that *APXD* indicator has a strong correlation with fault coverage. In addition, *APXD* simulation is 1900x, 63x, and 56x faster than serial, sampling, and parallel fault simulation, respectively. Exploiting *APXD* indicator instead of fault coverage, in a pruning-based test generation method, leads to about 700x, 24.2x, and 18.4x speedup, respectively compared to pruning based methods that use serial, sampling, or parallel fault simulation for test pattern evaluation, at fault coverage of 80%. Speedup at fault coverage of 95% is about 111.3x, 11.1, and 3.6x, respectively. While, the use of *APXD* indicator instead of fault coverage increases the number of test vectors by 2% at most, confirming the efficiency of *APXD* indicator compared with probabilistic and statistical approximate indicators.

Keywords

Approximate fault simulation, Test pattern generation, Probabilistic fault simulation, Fault sampling.

1. Introduction

Rapid technology scaling, emergence of complex digital designs, and their widespread use in critical domains have made accurate and rapid testing of digital systems more important every day. Digital system testing requires generation of an efficient set of test patterns so that it can detect manufacturing defects in the shortest possible time and with the highest level of confidence. Automatic test pattern generation (*ATPG*) methods can be categorized into two groups: 1) deterministic methods, 2) simulation-based methods. In deterministic test generation methods, each time, one fault is targeted and by using analytical methods, appropriate test vector(s) are generated to detect that fault. Fault coverage (*FC*) is high in deterministic methods, but they have a high time complexity. In simulation-based *ATPG* methods, a specific fault is not targeted. Instead, a set of test patterns is generated by random or pseudorandom approaches, then these test patterns are evaluated in terms of their fault coverage, and finally the best test patterns are determined and inserted in the final test set [1-2]. Simulation-based approaches are more scalable compared with deterministic methods. In addition, these methods can be used to generate software-based self-test (*SBST*) routines for processor components. It should be noted that due to functional and timing constraints, generation of *SBST* routines for processor components using deterministic test generation methods is not straightforward [3].

Inspired from software test data generation approaches [4-5], metaheuristic techniques can extensively be exploited in simulation-based *ATPG* methods to enhance test quality and speedup test generation process.

Metaheuristic approaches based on Genetic algorithm (*GA*) are commonly used for automatic test pattern generation. In [6] a GA-based test pattern generation approach is presented, which uses the amount of signal activity as the fitness function. In another study [7], *FC* of test patterns is used as the fitness function of genetic algorithm. Consequently, fault coverage is improved, but time of test generation increases. In [8], by generating a good initial population for *GA* using *D-algorithm*, an acceptable fault coverage in a shorter time is achieved, compared with similar methods. In [9], by providing diversity in the population using Immune concepts, the volume of generated test set is deceased. The GA-based *ATPG* approaches are used for generating test for reversible circuits [10], diagnostic test for transition faults [11], and for multiple faults models [12].

Several approaches use *ACO* [13], and *PSO* [14-16] for test generation. These methods have shorter generation time and lower test quality compared with GA-based *ATPG* methods.

In simulation-based *ATPG* methods, fault coverage is commonly used to evaluate test patterns. Fault coverage is a reliable and accurate indicator, but it requires fault simulation, which is very time-consuming. Numerous studies have been conducted to accelerate fault simulation. These studies can be classified into several categories: 1) Methods that use hardware accelerators and *FPGA*s, 2) Methods that use multicore and manycore processors, 3) Methods that use *GPU*s, 4) Mixed-level and hierarchical simulation method 5) Approximate and statistical methods.

Some researchers have used reconfigurable hardware to accelerate fault simulation. In [17], by embedding extra hardware, it is possible to emulate a fault in the circuit. In this study, fault injection is done dynamically using a shift register. The number of bits in this shift register is equal to the number of injectable faults. Adjusting each bit of this shift register to one will activate a fault in the circuit. The most important limitation of this method is that as the number of faults increases, the overhead of extra hardware increases. In [18], fault injection is performed by partial reconfiguration of *FPGA*, and by adjusting *LUT* reconfiguration vectors using small binary files. One of disadvantages of this method is the need to reconfigure the *FPGA* with each fault injection, although this reconfiguration is partial. In addition, fault injection is performed at *LUT* level, which is not exactly consistent with the gate-level fault injection. Another research [19] also uses the idea of partial reconfiguration, but to improve performance, reconfiguration is done using an embedded processor. Hardware methods used for acceleration of fault simulation have limited application and cannot replace the software methods of fault simulation, especially in simulation-based *ATPG* approaches.

Other research has used the processing power of multicore and manycore processors to speed up fault simulation algorithms. In [20], a fault simulator is implemented on *Intel's Single-Chip Cloud Computer*, which is able to balance load distribution on the processing cores during the runtime. This fault simulator uses message passing buffers to exchange data between cores. This fault simulator is 45x faster than serial fault simulation. In another study [21], a parallel fault simulator is implemented on the 20-core *Intel Xeon Processor* with the aim of speedup scalability. Implementation is done using *C++* and *OpenMP*. The results of this study show an average acceleration of 51x. Experimental results show that as the number of cores increases, the acceleration increases monotonically.

Another way to speed up fault simulation is to use *GPU*s. In [22], *GPU* processing power is used for fault simulation. In this method, block parallelism, fault parallelism, and pattern parallelism are used simultaneously. In [23], a fault simulator has been implemented on a *GPU* to create fault dictionary and enable fault diagnosis. In this study, pattern-parallelism and fault-parallelism have been used, and on average, 8x speedup is achieved compared to *CPU* implementation. In another study [24] a switch-level fault simulator is implemented on a *GPU*. The purpose of this fault simulator is to accurately examine the behaviour of *CMOS* cells under parametric faults and process variations, with the aim of test validation. This fault simulator exploits parallelism at the level of cell, stimuli, fault, and circuit instances. This simulator is able to achieve 243x speedup compared to gate-level timing simulation.

Recently, several researchers proposed mixed-level and hierarchical fault simulation. They use a mixed-model of a circuit consisting of high-level and gate-level components. Faults are injected in gate-level components and a pre and post synthesis co-simulation is performed [25-27]. Although these methods can significantly accelerate fault simulation, but cannot simulate general circuits, and are specifically designed and implemented for a predetermined class of digital circuits such as processors, or neural networks.

Another way to deal with time complexity in fault simulation is to use approximate or statistical methods. In [28], instead of accurately calculating fault coverage of a test set, the confidence level that fault coverage is higher than FC_{min} is calculated by using a fault sampling method. In another study [29], instead of using the usual algorithms for fault simulation, local fault simulation is used to estimate fault coverage. Experimental results of this study show that their proposed method is about ten times faster than the exact fault simulation method, but in comparison with fault sampling method (which is a fault simulation on a random subset of total faults), it is slower, although it is more accurate. In another study [30], a merit indicator that is based on probabilistic circuit analysis approach, is proposed for test pattern evaluation. This probabilistic indicator has an acceptable correlation with *FC* in most circuits, and can be calculated quickly.

In this paper, we propose an approximate indicator to substitute fault coverage in test pattern evaluation, which is more efficient than the previous ones. The rest of this article is organized in this way. In Section 2, problem statement and our contributions are presented. In Section 3, background concepts are introduced. In this section two approximate fault simulation methods, i.e., sampling fault simulation, and probabilistic fault simulation are briefly presented. Additionally, the concept of correlation coefficient and *sample-to-sample variability*, are explained. In Section 4, the proposed merit indicator, *APXD*, and the way it is calculated are investigated. Besides, a straightforward method for pruning-based test generation by use of approximate indicators, is provided. In Section 5, *APXD* indicator is evaluated in terms of correlation with fault coverage, and execution time, and then, it is compared with sampling and probabilistic fault simulation approaches. Moreover, in this section, the effect of using *APXD* as a substitute for fault coverage, in a pruning-based test generation algorithm, is examined in terms of test generation time and quality of test. Finally, in Section 6, we conclude the paper. In addition, a list of symbols and notations used in this paper is presented in the appendix.

2. Problem statement and our contributions

Simulation-based test pattern generation methods are promising solutions for *ATPG* of digital circuits. In this approach, random or pseudo random test patterns are generated on the fly. Traditionally, these test patterns are evaluated in terms of their fault coverage, and those with the highest coverage are selected and inserted in the final test set.

Test pattern evaluation according to their fault coverage needs numerous runs of fault simulation that is very time consuming. A promising solution to this challenge is using an approximate measure instead of fault coverage. Several approximate and statistical measures are proposed in the literature [28-30]. In this paper we propose an approximate measure for test pattern evaluation that can be calculated much faster than fault coverage, and still has a strong correlation with fault coverage. This measure, approximately counts the number of faults that each test pattern can detect. We call this approximate indicator as *APXD* (Approximate number of detected faults). Our evaluation shows that *APXD* outperforms previous approximate and statistical measures, in terms of accuracy and speed. To show effectiveness of *APXD*, we have implemented a simulation based test generation method that exploits *APXD* instead of fault coverage for test pattern evaluation. Our evaluations show that using *APXD* indicator instead of *FC*, in test generation algorithms, leads to a significant speedup with a negligible effect on the number of test vectors in the generated test set. Details of *APXD*, and our evaluations are presented in section 4 and Section 5.

3. Background

One way to tackle time complexity in calculation of fault coverage is to use approximate methods to estimate fault coverage, which can be beneficial especially in simulation-based *ATPG* methods. In simulation-based *ATPG* methods, several test patterns are generated using pure random or metaheuristic methods. Then, these test patterns are evaluated according to their *FC*, and the best ones are determined and included in the test set. Due to the inherent uncertainty that exists in generation and selection of test patterns in simulation-based *ATPG* methods, instead of *FC*, which is an exact but time-consuming indicator, an approximate and fast calculatable measure can be used to rate and choose more efficient test patterns. Two approximate methods for estimating fault coverage are sampling fault simulation [28, 31] and probabilistic fault simulation [30].

In sampling fault Simulation, instead of the fault simulation being performed for all faults in the fault list, a subset of faults, which we call sample set or sample, is selected and fault simulation is done only for this subset. By performing sampling fault simulation for a test pattern, a value called sampling fault coverage (*SMP_FC*) will be obtained, which is an approximation for the fault coverage of that test pattern. The smaller the sample size, the lower the accuracy of this approximation, but the higher the speedup.

In probabilistic fault simulation, circuit analysis is performed using probabilistic circuit analysis methods [30, 32]. Probabilistic analysis provides the possibility of simultaneous injection of faults. Traditionally in single stuck-at fault model, faults are injected one by one and one round of fault simulation is performed for each fault. In contrast, in probabilistic simulation method, a test pattern is applied to the circuit inputs, and all faults are injected simultaneously into the circuit according to the probabilistic fault injection rules. Then, by performing only one round of probabilistic simulation, effect of all faults are cumulated in probabilistic values of the primary outputs. Finally, according to probabilistic values of primary outputs, a measure called *PMI* (Probabilistic Merit Indicator) is calculated for each test pattern. The higher the *PMI* value for a test pattern, the higher the probability that the test pattern can detect a large number of faults.

Efficiency of each approximate indicator, that is used to substitute fault coverage, depends on two factors: 1) the speedup obtained from the approximate indicator, 2) the accuracy of that approximate indicator in evaluating test patterns, and selecting a pattern with a higher fault coverage. Equation (1) can be used to calculate speedup. In this equation, $t_i(fc)$ is the time required to calculate the exact fault coverage of the i^{th} test pattern, and $t_i(apx)$ is the time needed for calculation of the approximate indicator for the same test pattern. It should be noted that test patterns are generated randomly.

$$Speedup = \frac{\sum_i t_i(fc)}{\sum_i t_i(apx)} \qquad (1)$$

To check accuracy of an approximate indicator, the correlation coefficient of that indicator with fault coverage can be considered. For this purpose, a random set of *n* test patterns is generated. For each test pattern in this random set, fault coverage and the desired approximate indicator are calculated (assuming, the test pattern is applied to primary inputs of a benchmark circuit). In this way, *n* values will be obtained for fault coverage, and other *n* values for the approximate indicator. It is now possible to calculate correlation coefficient between the approximate indicator and fault coverage, using statistical methods, and by considering these two *n*-value vectors.

Two important methods for calculating correlation coefficient are *Spearman's rank correlation coefficient*, and *Pearson's correlation coefficient* [33]. It should be noted that selecting suitable type of correlation coefficient depends on the type of variables. When there is at least one ordinal scale variable, *Spearman's coefficient* is a better choice [33]. Equation (2) shows how to calculate *Spearman's correlation coefficient*. In this equation, *n* represents the number of data and d_i is difference in the order of variables.

$$corr_s = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \qquad (2)$$

Another point to consider when calculating the correlation coefficient between two variables is the value of *n*, i.e. the number of samples, which we call sample size. In our case, the sample size means the number of test patterns for which fault coverage and the approximate indicator are calculated to determine correlation coefficient. One important question is how sample size should be selected, that the correlation coefficient obtained from that sample has an acceptable correspondence with the case where all possible test patterns for that circuit are used to calculate the correlation coefficient. In response, the sample size should be chosen so that *sample-to-sample variability* is small. *Sample-to-sample variability* shows the maximum difference between a desired parameter (such as correlation coefficient calculated for a sample set) and the mean value of that parameter, if we repeat the experiment several times. Equation (3) shows how to calculate *sample-to-sample variability*. In this equation p_i is the value of a desired parameter calculated for the i^{th} sample set, and p_{avg} is the average value of all p_i values obtained from various sample sets.

$$S2SV = \frac{Max(p_i - p_{avg})}{p_{avg}} \qquad (3)$$

4. Proposed approximate indicator for quick evaluation of test patterns

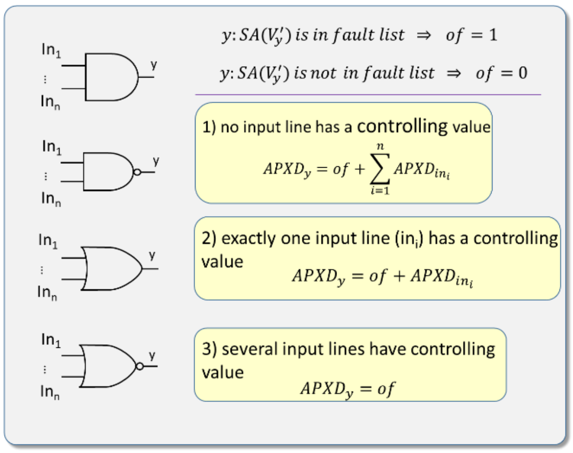
In this section, we first propose an approximate indicator for quick assessment of test patterns, called *APXD* (*approximate number of detected faults*). Then, in order to evaluate *APXD*, a straightforward test generation method, which exploits *APXD* for test pattern pruning is suggested.

4.1. APXD Approximate Indicator

APXD indicator (*approximate number of detected faults*) is a measure representing approximate number of faults that a test pattern can detect, considering all or a subset of single stuck-at faults in a circuit. This indicator can be calculated much faster than traditional fault coverage measure, and therefore its use for evaluation of candidate test patterns will significantly speed up simulation-based *ATPG* algorithms.

In *APXD* simulation, two values are assigned to each line (*L*) of a circuit. A logical value, which we denote by V_L, and an approximate indicator, which we call $APXD_L$. If *L* is output of a gate, V_L is determined by logical calculations on logical values of the gate inputs. $APXD_L$ represents the number of faults that are activated by a test pattern, and their effect propagate to Line *L*. In *APXD* simulation, circuit analysis starts from the primary inputs (*PI*), and by passing each gate, *V* and *APXD* for the output of each gate is calculated. Simulation continues until *V* and *APXD* for all primary outputs (*PO*) are determined. In more detail, a test pattern is applied to the primary inputs (*PI*) of the circuit. V_{PI} is then determined for all primary inputs directly according to the bit values of the test pattern. After that, $APXD_{PI}$ is calculated for each primary input according to the bit values of the test pattern, and considering the faults in the collapsed fault list. In fact, if fault "*PI stuck-at* (V'_{PI})", which we display as *PI: SA*(V'_{PI}), is in the fault list then $APXD_{PI} = 1$ and otherwise $APXD_{PI} = 0$. After determining *V* and *APXD* values for the primary inputs, the circuit analysis continues by passing through the gates and moving towards the primary outputs of the circuit. For each gate whose *V* and *APXD* of its inputs are determined, *V*, and *APXD* of the gate output can be calculated.

Calculation of *APXD* for output of a gate depends on the type of that gate and the logical values of inputs of that gate. Fig. 1 summarizes how *APXD* indicator is calculated for outputs of *AND*, *NAND*, *OR*, and *NOR* gates. As an example, let us look at *AND* gate. Suppose that all inputs of *AND* gate have a non-controlling value (controlling value for *AND* gate is 0). In this case, any fault whose effect reaches one of the gate inputs will propagate to the gate output. If we assume that each fault propagates to at most one input of a gate (meaning that reconvergent fanouts of the circuit are ignored), we can conclude that the number of faults that propagate to the output of the *AND* gate is equal to the sum of the faults propagated to inputs of the *AND* gate. This means that in this case, *APXD* of the output can be calculated by adding *APXD* of all inputs of the *AND* gate. However, it should be noted that fault *y: SA*(V'_y) will also be activated at the output of the gate (*y* is output line of the *AND* gate). Therefore, if this fault is in the faults list, $APXD_y$ should be increased by one.



In_1
In_n
y
$y: SA(V'_y)$ is in fault list $\Rightarrow$ $of = 1$
$y: SA(V'_y)$ is not in fault list $\Rightarrow$ $of = 0$
1) no input line has a controlling value
$APXD_y = of + \sum_{i=1}^{n} APXD_{in_i}$
2) exactly one input line (in_i) has a controlling value
$APXD_y = of + APXD_{in_i}$
3) several input lines have controlling value
$APXD_y = of$

Fig. 1. Calculating *APXD* indicator for *AND, NAND, OR*, and *NOR*

The second mode for the *AND* gate is when exactly one of the gate inputs (i^{th} input) has a controlling value. In this case, any fault whose effect reaches other inputs of the gate will be blocked due to the controlling value on the i^{th} input, and its effect will not propagate to the gate output. Therefore, only faults propagated to the i^{th} input pass through the gate, and propagate to the gate output. Therefore, in this case, *APXD* of the output is equal to the *APXD* of the i^{th} input. Of course, the fault activated in the gate output must be considered, if it is in the fault list.

The third case is when more than one of the gate inputs have a controlling value. In this case, no fault can pass through the gate, because there is always an input with a controlling value that blocks the effect of all faults. Fig. 1 summarizes *APXD* calculation rules for *AND*, *NAND*, *OR*, and *NOR* Gates. The rules are quite similar for these gates except that the controlling value for *OR* and NOR gates is 1 and for *AND*, and *NAND* gates is 0.

Fig. 2 shows how to calculate *APXD* for output of *NOT* and *BUF*. Any fault whose effect reaches the input of *NOT* and *BUF*, passes through these primitives, and propagates to the output. Therefore, *APXD* of output is equal to *APXD* of input of these primitives.

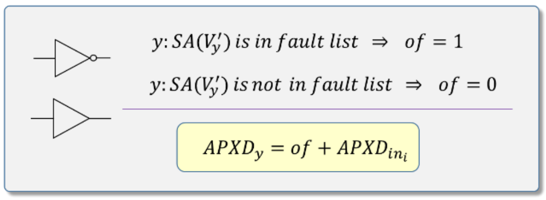

$y: SA(V_y') \text{ is in fault list} \Rightarrow of = 1$
$y: SA(V_y') \text{ is not in fault list} \Rightarrow of = 0$
$APXD_y = of + APXD_{in_i}$

Fig. 2. Calculating *APXD* indicator for *NOT*, and *BUF*

Fig. 3 shows the rules of *APXD* calculation for *FANOUT*, *XOR*, and *XNOR*. In the case of a *FANOUT*, when a fault reaches the stem, that fault propagate to all branches. Therefore, having *APXD* of the stem, *APXD* of all branches can be obtained. In the case of *XOR* and *XNOR*, we can say that any fault that reaches an input of these gates, propagates to the output, if we ignore reconvergent fanouts in the circuit for simplification. Therefore, *APXD* of output will be equal to the sum of *APXD* of all inputs in an *XOR*, or *XNOR* gate. However, we emphasize again that the fault activated at the gate output must also be considered in all cases.

After performing one round of *APXD* simulation, *APXD* indicator will be obtained for all primary outputs of the circuit. The higher the *APXD* indicator in more circuit *POs*, the more detected faults by the test pattern applied to the primary inputs of the circuit is expected. Based on this intuition, we propose Equation (4) as an approximate indicator to determine the efficiency of a test pattern in identifying more faults.

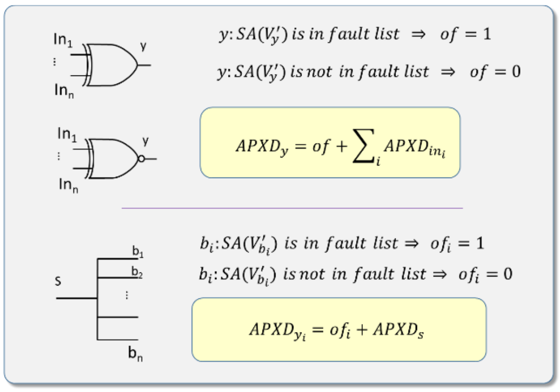

In_1
In_n
y
$y: SA(V_y') \text{ is in fault list} \Rightarrow of = 1$
$y: SA(V_y') \text{ is not in fault list} \Rightarrow of = 0$
$APXD_y = of + \sum_i APXD_{in_i}$
$b_i: SA(V_{b_i}') \text{ is in fault list} \Rightarrow of_i = 1$
$b_i: SA(V_{b_i}') \text{ is not in fault list} \Rightarrow of_i = 0$
$APXD_{y_i} = of_i + APXD_s$
S
b_1
b_2
b_n

Fig. 3. Calculating *APXD* indicator for *XOR, XNOR*, and *FANOUT*

In this Equation, $APXD_{TP}$ is *APXD* indicator for test pattern *TP*, and $APXD_{PO_i}$ is *APXD* indicator for the primary output PO_i. In Section 5, accuracy of *APXD* indicator in detecting high efficient test patterns is discussed.

$$APXD_{TP} = \sum_i APXD_{PO_i} \tag{4}$$

4.2. *Time Complexity of APXD calculation*

The advantage of *APXD* is the higher speed in calculation of *APXD* compared with *FC*. In serial fault simulation, each time, one fault is injected into the circuit, and by performing one round of simulation for each fault, it is determined whether that fault is detected by the test pattern. Therefore, if we show the number of faults as *f*, and the number of primitive gates in the netlist of a circuit as *g*, in serial fault simulation, *f* rounds of fault simulation is needed to evaluate a test pattern, and each round of simulation needs *g* operations. Equation (5) shows time complexity of serial fault simulation.

$$O(Serial\ F.S.) = O(f \times g) \tag{5}$$

In parallel fault simulation, instead of a separate simulation for each fault, a group of faults is checked simultaneously in one round of parallel fault simulation. The number of faults in a group depends on the word length of the processor. If we show the word length of the processor as *wl* (a constant value), in parallel fault simulation, *f / wl* rounds of parallel fault simulation is required to check *f* faults, each round consisting *g* operations. Equation (6) shows time complexity of parallel fault simulation.

$$O(Parallel\ F.S.) = O(f \times g) \tag{6}$$

In *APXD* simulation, a test pattern is applied to the primary inputs of a circuit, and by performing only one round of *APXD* simulation (one round of *g* operations), the number of faults that this test pattern can detect is calculated approximately. Therefore, *APXD* fault simulation is expected to be much faster than serial and parallel fault simulation methods, and this advantage increases in circuits with higher number of faults. Equation (7) shows time complexity of *APXD* fault simulation.

$$O(APXD) = O(g) \tag{7}$$

4.3. *Test Generation Based on Approximate Indicators*

As mentioned, *APXD* indicator provides the capability of quick test pattern evaluation and can lead to speed up in simulation-based *ATPG* methods. Fig. 4 shows *APXD*_TG test generation method that uses *APXD* approximate indicator to prune random test patterns, and select more efficient ones to insert in the final test set. In *APXD_TG*, a set of random test patterns is initially generated and inserted in a set called *curSet* (Fig. 4, stage 02). The number of these test patterns is denoted by parameter *setSize*. *APXD* indicator is then calculated for all test patterns in *curSet*, and a test pattern with the highest *APXD* value is selected (Fig. 4, stage 03, and 04). We call this test pattern *bestTP*. In the next step, *bestTP* is applied to the primary inputs of the circuit, and then parallel fault simulation is performed on the circuit (Fig. 4, stage 06). After parallel fault simulation, the number of faults detected by *bestTP* (denoted by *nNewDet*) is determined. If the number of new faults identified by *bestTP* exceeds our expected level, this test pattern is accepted and inserted in the final test set (Fig. 4, stage

08). The expectation level is adjusted by a parameter called *expDet*. The process of generating random test patterns and evaluating and accepting them continues until the generated test set provides our desired fault coverage (*finalFC*), or the algorithm reaches a point where, after several consecutive iterations, no progress is made in detecting new faults. In *APXD_TG* algorithm, *setSize* and *expDet* parameters can be used to adjust level of severity in accepting new test vectors. As the value of these parameters increases, test quality improves, but test generation time increases.

```
Algorithm: APXD_TG
Input:     netlist of a circuit
Output:    testset for the input circuit
Parameter: maxItt (maximum number of iterations)
Parameter: expDet (expected number of detected faults)
Parameter: finalFC (a desired value for fault coverage)
Parameter: setSize (number of random test patterns)
00: START
01: itt = 0; FC = 0;
02: Generate a set of random test patterns and insert them in a
      set called curSet (the number of random test patterns is
      equal to parameter setSize)
03: Calculate APXD for all test patterns in curSet.
04: Find a test pattern in curSet with the highest value of APXD
      (name it bestTP).
05: Apply bestTP to the primary inputs (PIs) of the input circuit.
06: Perform parallel fault simulation on the circuit and find the
      number of faults detected by bestTP (denoted by nNewDet).
07: IF (nNewDet < expDet)
        GOTO stage 11
08: Accept bestTP as a good test pattern and insert it in the final
      testset
09: drop all faults detected by bestTP from the fault list
10: update FC (fault coverage of all accepted test patterns)
11: IF (FC < finalFC) AND (i < maxItt)
        GOTO stage 02
12: END
```

Fig. 4. *APXD_TG* test generation algorithm

As mentioned, in *APXD_TG* algorithm, evaluation and pruning of test patterns are based on *APXD* indicator. Instead of *APXD*, the exact indicator, fault coverage (*FC*), or approximate indicators such as *PMI* (approximate indicator obtained from probabilistic fault simulation [30]), and *SMP_FC* (approximate indicator obtained from sampling fault simulation), can be used to select test patterns. In Section 5, the results of test generation based on *APXD* indicator, in terms of test quality and test generation time, are compared with the results of test generation based on other merit indicators, *FC*, *PMI*, and *SMP_FC*, and its superiority is shown.

Another point to note is that *APXD_TG* is a simple test generation algorithm based on the concept of test pattern pruning. In this algorithm, test pattern generation is performed pure randomly. The results of this algorithm, presented in Section 5, confirm the efficiency of *APXD* indicator in identification of efficient test patterns.

However, it should be emphasized that the use of metaheuristic concepts in generation of pseudo random test patterns can lead to improved test quality. For example, *APXD* can be used as a fitness function in a simulation-based test generation algorithm based on Genetic Algorithm (*GA*). In this approach, initially, several test patterns can be generated in a random fashion or using a deterministic test generation method to produce a good initial population for the *GA*-based solution. Then, fitness of each test pattern of the current generation is evaluated according to their *APXD* measure, and more efficient test patterns are selected stochastically using a proper selection mechanism such as Roulette wheel selection. The selected test patterns are used to generate the next generation by proper mutation and crossover operations. The process is terminated when a desirable fault coverage is achieved. We will work on this concept in a future work.

5. Experimental results

In this section, the proposed approximate indicator, *APXD*, is evaluated in terms of simulation speed, and accuracy. For this purpose, *APXD* fault simulation time is compared with two exact methods, serial fault simulation, and parallel fault simulation as well as two approximate methods, probabilistic fault simulation [30], and sampling fault simulation that were introduced in Section 3. Besides, to evaluate the accuracy of *APXD* in finding efficient test patterns, the correlation coefficient between *APXD* indicator and fault coverage is calculated in different circuits and is compared with the correlation coefficient of other approximate indicators. Experiments have been performed on several circuits of *ISCAS 85* benchmarks.

In order to be able to evaluate the proposed indicator (*APXD*) and compare it with the existing methods, we have designed and implemented a novel fault simulator with the capability of performing various exact and approximate fault simulations. We call this simulation engine, *LPSim* (Logical-Probabilistic) simulator. *LPSim* is implemented in *C++*, and is based on an object-oriented intermediate format called *PLEX* (Probabilistic and Logical Executable Model). *LPSim* allows logical simulation, and exact fault simulation in serial and parallel form. It provides the capability of probabilistic, sampling, and also *APXD* fault simulation. *LPSim* is easily extendable and various test methods can rapidly be developed and evaluated on this platform.

In all experiments related to calculating correlation coefficients, sample test sets include 2000 random test vectors. Each experiment is repeated for ten different sample test sets, and the results are the average of 10 different runs. Also, to show that sample sets are of sufficient size, maximum *sample-to-sample variation* is calculated in each experiment.

Table I shows correlation coefficient between the proposed approximate indicator, i.e. *APXD*, and fault coverage. Besides, the correlation coefficient of the probabilistic indicator (*PRB*) and approximate sampling indicator (*SMP*) with fault coverage are presented in this table. Results for *SMP* indicator are presented in three sampling rates of 3%, 5%, and 10%.

The results in Table I show that *APXD* indicator has a strong correlation with fault coverage, in all circuits. Therefore, it is an excellent alternative to fault coverage. The results also confirm that in all circuits, the correlation coefficient of *APXD* is higher than the correlation coefficient of *PRB* and *SMP* indicators. Another point is that the correlation coefficient of *APXD* indicator is in the range of strong correlation for all circuits while, the correlation coefficient of the other approximate indicators is sometimes in the range of moderate or even weak correlation.

Fig. 5 shows the maximum *sample-to-sample variation* in experiments of correlation coefficient. As we can see, the value of this parameter in experiments for all circuits is at most about 4%, and this confirms that the sample size selected to calculate the correlation coefficients (which is equal to 2000) is appropriate and the results are reasonably accurate.

Table I. Correlation coefficient between approximate indicators and exact fault coverage

Method	APXD	PRB [30]	SMP_3%	SMP_5%	SMP_10%
C432	0.96	0.83	0.48	0.51	0.75
C499	0.95	0.93	0.54	0.86	0.88
C880	0.93	0.88	0.51	0.57	0.63
C1355	0.97	0.93	0.61	0.87	0.91
C1908	0.94	0.86	0.74	0.85	0.87
C3540	0.84	0.58	0.54	0.54	0.67
C5315	0.97	0.9	0.82	0.86	0.89
C6288	0.83	0.63	0.19	0.21	0.61
C7552	0.89	0.43	0.42	0.52	0.65
Avg.	0.92	0.77	0.54	0.64	0.76

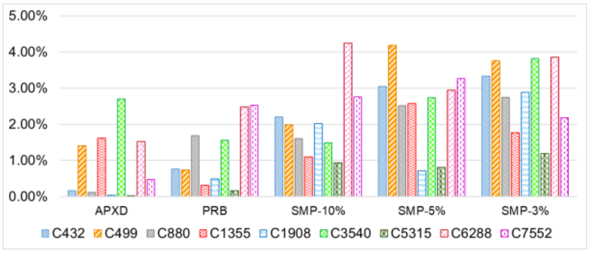


Fig. 5. Maximum Sample-to-Sample Variation

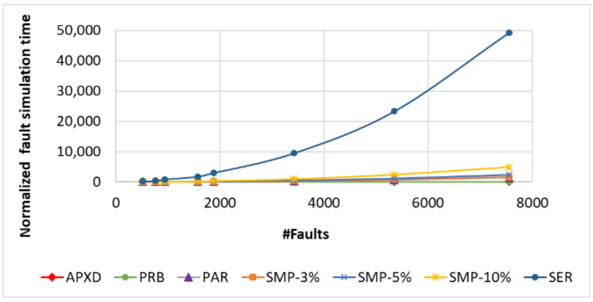


Fig. 6. Normalized fault simulation time

In the case of approximate indicators, another parameter that is important is the time required to calculate that indicator. Fig. 6 demonstrates the normalized execution time for different fault simulation methods. These diagrams show that execution time in *APXD* and *PRB* methods is much shorter than that in serial, parallel, and sampling methods.

Fig. 7 shows the same graphs on a logarithmic scale for a better comparison. As can be seen in this figure, the execution time in *APXD* method is close to *PRB* and slightly less. However, as mentioned before, correlation coefficient of *APXD* is significantly better than that of PRB method. This figure also shows that for larger circuits, *APXD* simulation is three orders of magnitude faster than serial fault simulation, and one order of magnitude faster than parallel fault simulation.

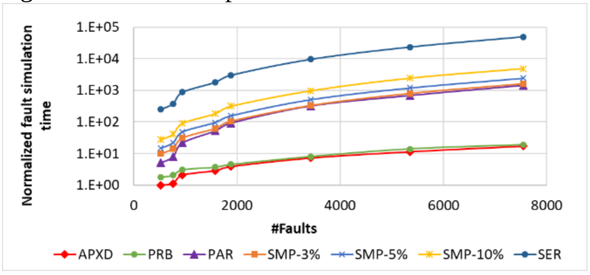


Fig. 7. Normalized fault simulation time in logarithmic scale

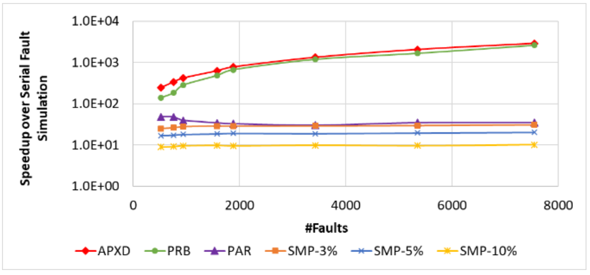


Fig. 8. Speedup in various fault simulation methods over serial fault simulation

Fig. 8 display the speedup resulting from different fault simulation methods compared to serial fault simulation in a logarithmic scale. The figure shows that the speedup in *APXD* method is slightly higher than the *PRB* method, but it is considerably higher than other methods. Additionally, in parallel and sampling fault simulation methods, as the number of faults increases, the speedup reaches a constant value, while in *APXD* method, as the number of faults grows, the resulting speedup also increases. This observation confirms that *APXD* approach is more scalable than parallel and sampling methods.

Fig. 9 shows average speedup in various fault simulation methods compared to serial fault simulation. It can be concluded, from Fig. 9, that *APXD* simulation is on average 1898x, 63x, 56x, 1.2x faster than serial, sampling (with sampling rate of 3%), parallel, and *PRB* fault simulation. It was discussed in Section 4.2 that *APXD* indicator can be used to accelerate simulation-based *ATPG* methods, and *APXD_TG* was proposed in this direction. In this section, the efficiency of *APXD_TG* is examined in terms of test generation time, and quality of test set that is generated. Then, it is compared with traditional methods that use fault coverage to evaluate and choose test patterns. Comparison is also done with other methods that use probabilistic [30] or statistical indicators for test pattern evaluation.

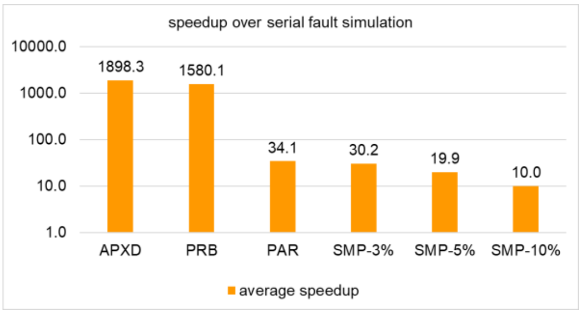


Fig. 9. Average speed up in various fault simulation methods over serial fault simulation

In all next tables and figures, different pruning-based test generation methods are denoted as *APXD_TG*, *PRB_TG*, and *SMP_TG*, *PAR_TG*, and *SER_TG*. *APXD_TG*, and PRB_TG are two pruning-based test generation methods that use *APXD* approximate indicator (see Section 4) and *PMI* (see Section 3) to evaluate candidate test patterns, respectively. *SMP_TG* is a pruning-based test generation method that uses approximate *SMP_FC* indicator (see Section 3) to evaluate test patterns. The number shown next to *SMP_TG* method implies the fault sampling rate. *SMP_TG_10%*, for example, shows that fault simulation is performed on 10% of all faults. Both *PAR_TG* and *SER_TG* methods use the exact merit indicator, i.e. fault coverage (*FC*), to evaluate efficiency of test patterns. In *PAR_TG* method, parallel fault simulation is performed to calculate *FC* for a test pattern, while *SER_TG* performs serial fault simulation.

Table II, III, and IV show the normalized execution time in different test generation methods at fault coverages of 80%, 90%, and 95%, respectively. The results show that the test generation time in *APXD_TG*, in all fault coverages, is much shorter than other methods, other than *PRB_TG*. The average test generation time in *PRB_TG* method is slightly higher than *APXD_TG*, but it is not much different. In order to better understand the results of these tables, Fig. 10 and Fig. 11 show the average speedup obtained from different test generation methods, in comparison with *SER_TG* (in logarithmic scale) and *PAR_TG* (in linear scale) methods, respectively. These results confirm that *APXD_TG* is about 695.9x, 419.4x, and 111.3x faster than *SER_TG*, respectively at fault coverages of 80%, 90%, and 95%. Besides, *APXD_TG* is 18.4, 11.5x, 3.6x faster than *PAR_TG*, at the above mentioned fault coverages.

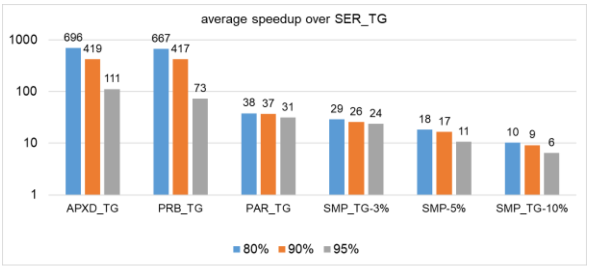


Fig. 10. Speedup in various test generation methods over *SER_TG*

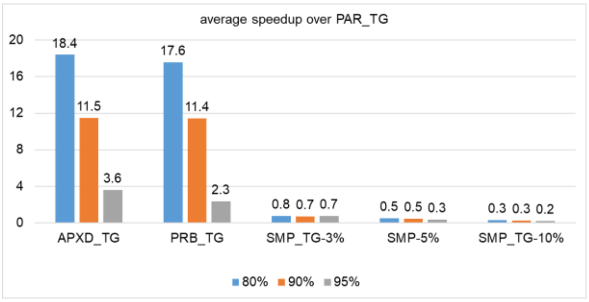


Fig. 11. Speedup in various test generation methods over *PAR_TG*

Table II. Normalized test generation time in fault coverage of 80%

Method	**C432**	**C499**	**C880**	**C1355**	**C1908**	**C3540**	**C5315**	**C6288**	**C7552**	**Avg.**
APXD_TG	3.0	1.0	3.7	3.2	8.1	33.5	25.7	8.0	42.7	14.3
PRB_TG [30]	3.3	1.0	3.5	2.5	6.4	39.8	27.5	8.9	41.6	14.9
SMP_TG_3%	11.5	4.7	23.6	27.2	74.5	491.0	691.0	334.0	1453.8	345.7
SMP_TG_5%	13.9	7.7	31.7	38.2	116.9	711.6	1412.6	436.8	2117.5	543.0
SMP_TG_10%	25.1	12.0	52.7	68.6	191.4	1433.9	2064.2	976.9	4001.5	980.7
PAR_TG	7.1	3.1	13.9	20.7	58.5	367.5	509.0	295.7	1088.0	262.6
SER_TG	190.7	109.8	453.3	691.8	1816.6	11835.3	18373.0	9900.6	46267.3	9959.8

Table III. Normalized test generation time in fault coverage of 90%

Method	**C432**	**C499**	**C880**	**C1355**	**C1908**	**C3540**	**C5315**	**C6288**	**C7552**	**Avg.**
APXD_TG	4.3	3.4	6.4	11.7	20.1	62.2	46.0	11.6	136.6	33.6
PRB_TG [30]	5.4	4.0	5.7	9.1	19.5	68.4	46.0	13.9	132.1	33.8
SMP_TG_3%	15.1	10.5	33.3	55.6	144.7	729.7	872.3	369.7	2728.0	551.0
SMP_TG_5%	18.4	14.1	43.5	75.5	204.4	1028.3	1773.1	499.7	4016.4	852.6
SMP_TG_10%	30.7	22.0	66.6	135.2	323.6	2321.7	2781.5	1069.9	7068.2	1535.5
PAR_TG	9.6	7.4	18.5	41.8	95.9	517.5	643.5	336.8	1801.7	385.9
SER_TG	217.5	169.9	537.5	1188.4	2748.5	15787.1	22745.0	11076.2	72377.4	14094.2

Table IV. Normalized test generation time in fault coverage of 95%

Method	C432	C499	C880	C1355	C1908	C3540	C5315	C6288	C7552	Avg.
APXD_TG	5.9	7.8	8.8	22.1	33.8	112.6	69.1	14.9	1124.9	155.5
PRB_TG [30]	5.8	7.6	8.5	19.0	125.6	1029.1	72.0	19.8	862.2	238.9
SMP_TG_3%	18.4	17.9	40.1	79.2	184.7	-	1067.2	399.9	-	200.8
SMP_TG_5%	21.8	19.8	53.4	106.4	258.0	1535.1	1978.4	544.8	10205.3	1635.9
SMP_TG_10%	34.6	30.6	78.5	179.9	407.8	2970.9	3213.8	1124.8	16169.6	2690.1
PAR_TG	11.8	13.4	22.6	60.0	122.7	644.6	730.7	358.5	3043.0	556.4
SER_TG	178.8	224.6	571.5	1499.1	3248.1	18540.0	25106.5	11683.6	94806.1	17317.6

Table V, VI, and VII show the number of test vectors produced by different test generation methods in faults coverage of 80%, 90%, and 95%, respectively. Column *TPR* (Test Pattern Ratio) in these tables has been calculated according to Equation (8). In this equation, nTV_{apx} is the number of test vectors in a final test set produced by a test generation method that exploits an approximate indicator for test vector evaluation, i.e. *APXD_TG*, *PRB_TG*, and *SMP_TG* methods. nTV_{FC} is the number of test vectors in a test set produced by a test generation method that uses fault coverage to prune test vectors, i.e. *SER_TG*, or *PAR_TG*.

$$TPR = \frac{nTV_{apx}}{nTV_{FC}} \qquad (8)$$

The results of Table V, VI, and VII make it possible to compare the accuracy and efficiency of test generation based on approximate indicators. The results show that *APXD_TG* method is significantly more efficient than other approximate methods, namely *PRB_TG* and *SMP_TG*. *APXD_TG*, with only 2%, 1%, and 2% increase in the number of test patterns (in three fault coverage of 80%, 90%, and 95%, respectively), is able to achieve a fault coverage equal to the coverage of *PAR_TG* and *SER_TG* methods.

6. Discussion and Conclusion

Simulation-based *ATPG* methods are interested because of their lower execution time. In these methods, fault coverage indicator is traditionally used to evaluate test patterns and eliminate patterns with lower efficiency. Although test patterns can accurately be evaluated by their fault coverage index, but calculation of fault coverage requires fault simulation, which is very time-consuming. Instead of fault coverage, approximate indicators can be used to evaluate test patterns. Calculation of approximate indicators is much faster than fault coverage.

In this paper, we propose an approximate indicator called *APXD* as a suitable alternative to fault coverage indicator. In *APXD* simulation, all faults are considered simultaneously. This means that by only one round of simulation for each test pattern, the number of faults that can be detected by that test pattern is calculated approximately. In *APXD* simulation, in addition to propagating logical values from inputs to output of gates, having the number of faults propagated to inputs of a gate, the number of faults whose effect propagate to the output of that gate is also calculated.

Table V. Test set size in fault coverage of 80%

Method	APXD_TG	PRB_TG [30]	SMP_TG_3%	SMP_TG_5%	SMP_TG_10%	PAR_TG	SER_TG
C432	13	15	18	16	17	13	13
C499	4	5	5	6	5	4	4
C880	9	11	14	13	11	9	9
C1355	7	7	8	8	7	7	7
C1908	13	14	16	18	15	13	13
C3540	27	30	36	34	30	25	25
C5315	14	19	19	18	16	14	14
C6288	4	6	5	4	5	4	4
C7552	16	20	21	20	18	16	16
TPR	1.02	1.21	1.35	1.3	1.18	1	1

Table VI. Test set size in fault coverage of 90%

Method	APXD_TG	PRB_TG [30]	SMP_TG_3%	SMP_TG_5%	SMP_TG_10%	PAR_TG	SER_TG
C432	19	25	28	27	26	19	19
C499	14	18	19	18	17	14	14
C880	16	18	27	25	19	15	15
C1355	26	26	31	31	29	26	26
C1908	33	41	50	50	41	33	33
C3540	59	62	80	74	71	60	60
C5315	26	33	33	32	32	25	25
C6288	6	8	7	6	7	6	6
C7552	53	56	73	72	58	51	51
TPR	1.01	1.15	1.4	1.35	1.2	1	1

Table VII. Test set size in fault coverage of 95%

Method	APXD_TG	PRB_TG [30]	SMP_TG_3%	SMP_TG_5%	SMP_TG_10%	PAR_TG	SER_TG
C432	26	37	40	38	36	26	26
C499	30	36	40	32	32	30	30
C880	22	27	39	40	30	22	22
C1355	49	54	58	60	54	49	49
C1908	56	69	82	83	71	56	56
C3540	105	106	-	152	129	101	101
C5315	40	53	61	53	54	38	38
C6288	8	15	10	9	9	8	8
C7552	154	167	-	183	171	151	151
TPR	1.02	1.17	1.44	1.35	1.22	1	1

Our experimental results demonstrate that *APXD* approximation is strongly correlated with fault coverage indicator. Compared to other approximate methods such as probabilistic and statistical methods, *APXD* indicator has a higher correlation coefficient with fault coverage. In terms of execution time, our experiments show that *APXD* indicator can be calculated about 1900x and 56x faster than fault coverage, using serial, and parallel fault simulation, respectively. Additionally, *APXD* simulation is about 63x faster than sampling fault simulation with a sampling rate of 3%. The short execution time and accuracy of *APXD* indicator confirms that this indicator is a good candidate to replace fault coverage indicator in simulation-based *ATPG* methods. Our experimental results show that in a pruning-based test generation method, the use of *APXD* indicator instead of fault coverage, in fault coverage of 80%, can lead to a speedup of about 700x, 24.2x, and 18.4x compared with serial, sampling, and parallel methods, respectively. Speedup values obtained in the 95% fault coverage are 111.3x, 11.1x, and 3.6x. Besides, using *APXD* indicator, instead of fault coverage, for test pattern pruning, leads to at most 2% increase in the final test set size.

It is worth mentioning the proposed test generation method, *APXD_TG*, is not adapted and evaluated for test generation of sequential circuits. Although the method can be adapted for sequential circuits, but this is not a big issue. Practically, sequential circuits are turned into combinational circuits with design for test techniques, and existing combinational test generation methods are applied to them.

Additionally, we have exploited the proposed approximate indicator, called *APXD*, in a pruning based test generation method to show its effectiveness. In this approach test patterns are generated in a pure random fashion and then evaluated and pruned according to their *APXD* value. Using metaheuristic approaches to generate more efficient pseudo random test patterns can improve the quality of the final test set as well as test generation time, which we will consider as our future work.

7. References

[1] E. O. Osimiry, R. Ubar, S. Kostin, and J. Raik, "A novel random approach to diagnostic test generation," in 2016 IEEE Nordic Circuits and Systems Conference (NORCAS), 2016, pp. 1-4.
[2] A. Kamran, M. S. Jahangiry, and Z. Navabi, "Merit based directed random test generation (MDRTG) scheme for combinational circuits," in 2010 East-West Design & Test Symposium (EWDTS), 2010, pp. 416-419.
[3] A. Kamran, "HASTI: hardware-assisted functional testing of embedded processors in idle times," IET Computers & Digital Techniques, vol. 13, no. 3, pp. 198-205, 2019.
[4] S. Esfandyari, V. Rafe, “A Hybrid solution for Software testing to minimum test suite generation using hill climbing and bat search algorithms”, Tabriz Journal of Electrical Engineering, vol. 46, no. 3, pp. 25-35, 2016 (in persion).
[5] M. M. Dejam Shahabi, S. E. Beheshtian, P. Badiei, R. Akbari, S. M. R. Moosavi, “Adapting Swarm Intelligence Based Methods for Test Data Generation”, Tabriz Journal of Electrical Engineering, vol. 51, no. 2, pp. 183-193, 2021.
[6] E. M. Rudnick, J. G. Holm, D. G. Saab, and J. H. Patel, "Application of simple genetic algorithms to sequential circuit test generation," in Proceedings of European Design and Test Conference EDAC-ETC-EUROASIC, 1994, pp. 40-45.
[7] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "A genetic algorithm framework for test generation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, no. 9, pp. 1034-1044, 1997.
[8] H. Harmanani and B. Karablieh, "A hybrid distributed test generation method using deterministic and genetic algorithms," in Fifth International Workshop on System-on-Chip for Real-Time Applications (IWSOC'05), 2005, pp. 317-322.
[9] M. Azimipour, M. R. Bonyadi, and M. Eshghi, "Using immune genetic algorithm in ATPG," Australian Journal of Basic and Applied Sciences, vol. 2, no. 4, pp. 920-928, 2008.
[10] A. N. Nagamani, S. N. Anuktha, N. Nanditha, and V. K. Agrawal, "A Genetic Algorithm-Based Heuristic Method for Test Set Generation in Reversible Circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 2, pp. 324-336, 2018.
[11] A. Bhar, S. Chattopadhyay, I. Sengupta, and R. Kapur, "GA based diagnostic test pattern generation for transition faults," in 2015 19th International Symposium on VLSI Design and Test, 2015, pp. 1-6.
[12] J. P. Anita and P. T. Vanathi, "Genetic algorithm based test pattern generation for multiple stuck-at faults and test power reduction in VLSI circuits," in 2014 International Conference on Electronics and Communication Systems (ICECS), 2014, pp. 1-6.
[13] R. Farah and H. Harmanani, "An Ant Colony Optimization approach for test pattern generation," 2008 Canadian Conference on Electrical and Computer Engineering, pp. 001397-001402, 2008.
[14] M. M. Alateeq and W. Pedrycz, "Analysis of optimization algorithms in automated test pattern generation for sequential circuits," in 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2017, pp. 1834-1839.
[15] G. Yuan-Liang and X. Wen-Bo, "Study on Automatic Test Generation of Digital Circuits Using

Particle Swarm Optimization," in 2011 10th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, 2011, pp. 324-328.
[16] Z. Jiali, Z. Lin, Y. Yun, N. Tianlin, Z. Long, and X. Xiaodong, "The Test Pattern Generation for Digital Integrated Circuits Based on CA-IA-PSO Algorithm," in 2015 Seventh International Conference on Measuring Technology and Mechatronics Automation, 2015, pp. 1316-1320.
[17] M. Santos, H. Braga, I. Teixeira, J. P. Teixeira, "Dynamic Fault Injection Optimization for FPGA-Based Harware Fault Simulation, " Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS), 2002, pp. 370-373.
[18] A. Parreira, J. P. Teixeira, A. Pantelimon, M. B. Santos, and J. T. de Sousa, "Fault Simulation Using Partially Reconfigurable Hardware," vol. 2778, pp. 839-848, 2003.
[19] L. Kafka and O. Novak, "FPGA-based fault simulator," in 2006 IEEE Design and Diagnostics of Electronic Circuits and systems, 2006, pp. 272-276.
[20] M. Haghbayan, S. Teräväinen, A. Rahmani, P. Liljeberg, and H. Tenhunen, "Adaptive fault simulation on many-core microprocessor systems," in 2015 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS), 2015, pp. 151-154.
[21] S. Hadjitheophanous, S. N. Neophytou, and M. K. Michael, "Scalable parallel fault simulation for shared-memory multiprocessor systems," in 2016 IEEE 34th VLSI Test Symposium (VTS), 2016, pp. 1-6.
[22] M. Li and M. S. Hsiao, "3-D Parallel Fault Simulation With GPGPU," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 10, pp. 1545-1555, 2011.
[23] M. Beckler and R. D. Blanton, "Fault simulation acceleration for TRAX dictionary construction using GPUs," in 2017 IEEE International Test Conference (ITC), 2017, pp. 1-9.
[24] E. Schneider and H. Wunderlich, "SWIFT: Switch-Level Fault Simulation on GPUs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 1, pp. 122-135, 2019.
[25] J. T. Xiao, T. S. Hsu, C. M. Fuchs, Y. T. Chang, J. J. Liou, and H. H. Chen, "An ISA-level Accurate Fault Simulator for System-level Fault Analysis," in 2020 IEEE 29th Asian Test Symposium (ATS, pp. 1-6), 2020.
[26] M. Karami, M. H. Haghbayan, M. Ebrahimi, A. Miele, H. Tenhunen, and J. Plosila, "Hierarchical Fault Simulation of Deep Neural Networks on Multi-Core Systems," in 2021 IEEE European Test Symposium (ETS), pp. 1-2, 2021.
[27] P. R. Maier, U. Sharif, D. Mueller-Gritschneder, and U. Schlichtmann, "Efficient Fault Injection for Embedded Systems: As Fast as Possible but as Accurate as Necessary," in 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS, pp. 119-122), 2018.
[28] F. M. Goncalves, M. B. Santos, I. C. Teixeira, and J. P. Teixeira, "Self-checking and fault tolerance quality assessment using fault sampling," in 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2002. DFT 2002. Proceedings., 2002, pp. 216-224.
[29] S. Mirkhani, J. A. Abraham, T. Vo, H. Jun, and B. Eklow, "FALCON: Rapid statistical fault coverage estimation for complex designs," in 2012 IEEE International Test Conference, 2012, pp. 1-10.
[30] M. Fooladi and A. Kamran, "Speed-Up in Test Methods Using Probabilistic Merit Indicators," Journal of Electronic Testing, vol. 36, no. 2, pp. 285-296, 2020/04/01 2020.
[31] S. A. Al-Arian and M. A. Al-Kharji, "Fault simulation and test generation by fault sampling techniques," in Proceedings 1992 IEEE International Conference on Computer Design: VLSI in Computers & Processors, 1992, pp. 365-368.
[32] G. Asadi and M. B. Tahoori, "An analytical approach for soft error rate estimation in digital circuits," in 2005 IEEE International Symposium on Circuits and Systems, 2005, pp. 2991-2994 Vol. 3.
[33] M. M. Mukaka, "Statistics corner: A guide to appropriate use of correlation coefficient in medical research," Malawi medical journal : the journal of Medical Association of Malawi, vol. 24, no. 3, pp. 69-71, 2012.

8. Appendix

Table VIII demonstrates a list of symbols and notations used in this paper.

Table VIII. List of Symbols and notations

Symbol	**Remarks**
ATPG	Automatic Test Pattern Generation
GA	Genetic Algorithm
FS	Fault Simulation
APXD	Approximate number of detected faults
TP	Test Pattern
FC	Fault Coverage
SMP_FC	sampling fault coverage
PMI	Probabilistic Merit Indicator
L	A line in a circuit netlist
V_L	Logical value of line L
$APXD_L$	*APXD* of line L
PI	Primary Input
PO	Primary Output
APXD_TG	Test generation based on *APXD* indicator
PRB_TG	Test generation based on *PMI*
SMP_TG	Test generation based on sampling
SER_TG	Test generation based on serial fault simulation
PAR_TG	Test generation based on parallel fault simulation