Test Pattern Generation for Approximate Circuits Based on Boolean Satisfiability

Anteneh Gebregiorgis and Mehdi B. Tahoori
Chair of Dependable Nano Computing (CDNC)
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
Email: {anteneh.gebregiorgis, mehdi.tahoori}@kit.edu

***Abstract*—Approximate computing has gained growing attention as it provides trade-off between output quality and computation effort for inherent error tolerant applications such as recognition, mining, and media processing applications. As a result, several approximate hardware designs have been proposed in order to harness the benefits of approximate computing. While these circuits are subjected to manufacturing defects and runtime failures, the testing methods should be aware of their approximate nature. In this paper, we propose an automatic test pattern generation methodology for approximate circuits based on boolean satisfiability, which is aware of output quality and approximable vs non-approximable faults. This allows us to significantly reduce the number of faults to be tested, and test time accordingly, without sacrificing the output quality or test coverage. Experimental results show that, the proposed approach can reduce the fault list by 2.85× on average while maintaining high fault coverage.**

I. Introduction

The recent decline in voltage scaling and performance improvement of technology scaling has led to the exploration of viable alternative computing paradigms so that the benefits of CMOS scaling is harnessed [1]. Among these, approximate computing, where output quality is traded-off for performance or energy saving, is regarded as one of the most promising technique for energy-efficient designs [2, 3]. Indeed, approximate computing proves itself as an energy-efficient approach for several applications including media processing, recognition, and mining tasks as they have inherent tolerance to inexactness in their output quality [1, 2]. To exploit the advantages of approximate computing, various approximate hardware designs have been proposed all the way from circuit level (e.g., adders and multipliers [4]) to processing unit such as CPU and accelerators [4, 5]. Moreover, the development of approximate benchmark suite platform has been investigated to provide more accurate performance and energy-efficiency evaluation of approximate hardware designs [6].

Similarly, logic and behavioral synthesis support has been explored in order to automate the generation and synthesis of approximate circuits [7]. However, the impact of manufacturing defects and runtime failures on output error rate of approximate circuits has been ignored to the mercy of fault tolerance [8]. In fact, these manufacturing defects and runtime failures can significantly increase the output error rate of approximate circuits to the extent of delivering unacceptable output quality [8–12]. As approximate computing is becoming a popular computing paradigm, and chips with approximate hardware component are fabricated, it has become imperative to have an effective testing flow for approximate circuits [8, 9].

In the conventional test flow of exact circuits, the test should have high fault coverage by successfully exciting and propagating all faults to any reachable output. Unlike exact circuit testing, approximate circuit testing should be aware of the approximation nature of the circuit by distinguishing *approximable* vs *non-approximable* outputs during fault excitement and propagation [13]. A circuit portion is considered as approximable iff a fault in that portion does not have an observable effect (i.e., output error is within the specified error margin) otherwise it is deemed as non-approximable. Thus, in comparison to the conventional test flow, approximate circuit testing comes with its own benefits and unique challenges. Approximate computing enables to reduce test time and costs as significant portion of the faults are propagated only to the approximable portion of the design leading to no observable impact on the output quality. Such faults are considered as *approximable faults* and can be ignored safely while the *non-approximable faults*, faults resulting in error tolerance margin violation, are crucial and must be detected. Moreover, the propagation of such faults to the outputs must be done in a way to violate the error tolerance margin, in order to detect the non-approximable faults.

In this paper, we propose a boolean satisfiability based test pattern generation methodology for approximate circuits. In the proposed approach, the approximate circuit testing problem is transformed into an instance of boolean satisfiability (SAT) problem and a SAT solver is used to identify non-approximable faults and generate their corresponding test patterns. Unlike the recent works which are either fully based on conventional ATPG [8] or augmented conventional ATPG [9], the proposed approach is fully aware of the error tolerance margin of an approximate circuit. Moreover, since the conventional ATPG tools prioritize shortest propagation paths of faults to the primary outputs, the solutions based on conventional ATPG can wrongly classify non-approximable faults as approximable or undetectable faults [14, 15]. To address this problem, our SAT based testing methodology inherently evaluates all propagation paths and converts the error tolerance margin into SAT formulation in order to classify true approximable and true undetectable faults correctly.

The effectiveness of the proposed approach is demonstrated by using simple and complex variants of approximate floating and fixed-point circuits. Results show that, the proposed approach can reduce the testable faults by 2.85× on average while maintaining high fault coverage. Moreover, the proposed approach can effectively detect all false-approximable and false-undetectable faults identified in previous techniques [9].

The rest of the paper is organized as follows: approximate circuit testing background and the related works are explained in Section II. Section III presents the proposed methodology followed by the experimental results in Section IV. Finally, Section V concludes the paper.

978-3-9819263-2-3/DATE19/©2019 EDAA

II. Background

A. Test aspects of approximate circuits

Since approximate circuits are designed to operate on applications with inherent error resiliency, some observable output errors can be safely ignored as long as the final output error is within the specified error tolerance margin. During fabrication and operation time, however, the circuits can be affected by manufacturing defects, environmental or aging induced faults. These faults can be propagated either to the approximable portion of the circuit without exceeding the error tolerance margin or show-up in the non-approximable portion leading to the violation of the specified error margin [8].

Therefore, any approximate circuit has to be tested in a way that all observable faults that could be propagated to the non-approximable portion of the circuit are detected by at least one test pattern [9]. Unlike exact circuit testing, an approximate circuit testing requires the generation of test patterns that are able to detect the violation of the specified error tolerance margin. Thus, an approximate circuit (AC) testing problem can be defined as follows: An approximate circuit is said to *pass* a test in the presence of fault, *f*, if and only if the output error of the faulty approximate circuit is within the specified error tolerance margin for all test patterns (T_P). This condition is stated mathematically in Equation 1.

$$AC_{test} = \begin{cases} \text{pass}, & \text{iff } \forall_{f \epsilon fault_{list}} |AC_i^R - E_i^R| \leq E_m \forall_i \ \epsilon \ T_p \\ \text{fail}, & \text{otherwise} \end{cases} \tag{1}$$

where fault_{list} is list of all possible faults, AC_i^R is AC's response for the i^{th} test pattern, E_i^R is the expected response of the i^{th} test pattern, E_m is the error tolerance margin specified by the designer and T_P is the set of test pattern.

Therefore, the error tolerance margin of approximate circuits should be considered and leveraged during test pattern generation phase in order to reduce the test pattern size and associated test cost of approximate circuits. Moreover, it is also necessary to correctly propagate the faults to non-approximable outputs to be able to observe the fault effects.

B. Related works

Majority of the recent works are mainly focused on energy-efficient design aspect of approximate computing [1–5]. Thus, the classical ATPG based test flow is indiscriminately applied to all circuits regardless of their expected level of exactness. As a result, significant test cost and time is being wasted to ensure high fault coverage by detecting maximum possible faults without considering the error tolerance capability of approximate circuits [12, 13].

A structural and functional analysis based approximation of circuit testing is presented in [12] and [13]. Although these works can serve as a good starting point for testing of approximate circuit, the analysis is used to identify and protect the vulnerable circuit elements only. Hence, these works does not consider the inherent error tolerance of the circuit and are applied to regular non-approximate circuits as they mainly focus on approximating the test process. Other important works on testing of approximate circuit are presented in [8, 10, 11]. The works in [8, 10, 11] mainly focus on a fully conventional ATPG based testing of approximate circuits by comparing the response of an approximate circuit to its exact (golden) version. Although these works are pioneers in approximate circuit testing, their effectiveness is limited to smaller circuit structures such as adders, as the test instance (DUT) has to include both golden and approximate variants.

A SAT based augmented ATPG technique is presented in [9]. The authors used SAT tool to pre-process and classify the faults into *acceptable* and *unacceptable* categories before applying ATPG. However, the fully and augmented conventional ATPG based techniques [8, 9, 11] have two main limitations. Firstly, the test pattern generation is not fully aware of the approximation margin as conventional ATPG tools mainly focus on propagating faults to any of the primary outputs and the comparison is done afterwards. Secondly, several detectable non-approximable faults can be wrongly classified as approximable (false approximable) by the ATPG tools. Since ATPG tool prioritize shortest propagation paths of faults to the primary outputs, it cannot be forced to check all possible propagation paths. Hence, faults which can propagate to multiple primary outputs are victims of this limitation and can be misclassified. The issue of false undetectable faults is also another limitation of such approaches.

III. Proposed SAT based approximate circuit testing methodology

A. Motivational example

Approximation aware testing of approximate circuits leads to significant test cost reduction and yield improvement as several manufacturing defects and runtime failures can be tolerated inherently. As a result, fabricated defective chips can be used as long as the faults are not propagated to the *non-approximable* portion of the circuit. To illustrate this and motivate the necessity of approximate circuit testing, we used a 4-bit approximate fixed-point adder and studied the impact of Stuck-At-Fault (SAF) on its accuracy. In this example, we choose fixed-point representation over integer as it has more approximation freedom with a relatively small error margin.

Fixed-point representation is one way of dealing with real numbers for designs without a dedicated floating point unit and it is widely used in different signal processing and gaming applications where performance is more important than precision, as fixed-point arithmetic is faster than floating point arithmetic [16]. In fixed-point notation, a real number is represented by allocating different bit sizes to the integer and fraction parts which are separated by binary point [16]. Hence, real numbers have the same binary representation as normal integers,; however, the fraction part (bits to the right of the binary point) will have negative weight (exponent). For example, in a 4-bit fixed-point representation with 2-bit integer and 2-bit fraction part allocation, the value 2.75 will represented as follows (Table I):

TABLE I. Fixed-point representation

Binary values →	1	0	.	1	1
Interpretation →	1×2^1	0×2^1	.	1×2^{-1}	1×2^{-2}

Depending on the requirement (range, and precision) the location of the binary point (allocated bits for integer and fraction part) can be changed. In the above example, the range is [0-3.75] with a precision of *0.25*.

Let us now consider the fault propagation and test aspect of the approximate 4-bit adder design given in Figure 1 by using SAF fault model with an error tolerance margin of 0.5. Based on Equation (1), a fault is deemed as approximable iff the output error of the approximate adder in the presence of the fault is not greater than 0.5 (the specified margin).

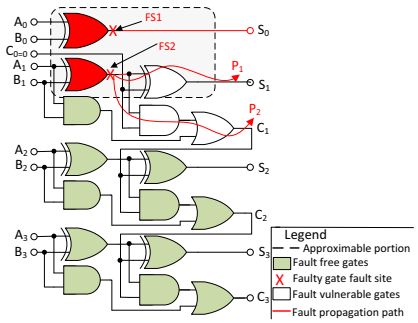


Fig. 1. Four-bit approximate fixed-point adder design

For simplicity let us assume the fault sites *FS1* and *FS2* given in Figure 1 have a Stuck-at-1 (SA1) fault. Since the fault at *FS1* has only one propagation path to the primary output, it is straightforward to determine if it is approximable or not. However, the fault at *FS2* has two propagation paths (P_1 and P_2 as indicated by the red line in Figure 1). Therefore, both propagation paths should be evaluated in order to determine whether the fault is approximable or not. Table II shows the truth table of the adder for golden (exact), fault free approximate and faulty approximate with faults at FS1 and FS2. Since the fault sites are at the lower output bits of the adder, the table presents the truth table of the lower input bits A_1A_0 and B_1B_0) only. Thus, the upper input bits (A_3A_2 and B_3B_2) have no impact on the fault sites and are assumed to be constant (e.g., we used $A_3A_2 \leftarrow$'10' and $B_3B_2 \leftarrow$'10').

The first column of Table II represents the two lower bits of the input operands and their decimal values (i.e., input operands in decimal notation). The second column shows the outputs of the exact adder. The fault free approximate adder outputs (Figure 1) are given in the third column and the accuracy degradations are indicated by the ($\pm\epsilon$). It should be noted that the worst case error of the fault free approximate adder is within the specified margin (0.5). The outputs of the approximate adder with FS1-SA1 are given in the fourth column. From the fourth column we can observe that the SA1 fault at fault site FS1 does not violate the error margin and hence it is approximable. Similarly, the outputs of FS2-SA1 through propagation paths P_1 and P_2 are given in the fifth and sixth columns. The SA1 at FS2 can be considered as approximable when it is propagated through path P_1, as the output errors are within the specified error margin (as shown in the fifth column of Table II). However, when the fault is propagated through path P_2, the output error violates the specified error margin (e.g., +1.0 as shown in the sixth column). Therefore, the approximation-aware test flow should detect the SA1 fault at FS2 and classify it as a non-approximable fault and generate an appropriate test pattern to detect it. If conventional ATPG is used, such faults are most likely to be misclassified as the ATPG tool propagates them through shortest paths (similar to P_1) and the tool cannot be forced to evaluate alternative routes (propagation paths such as P_2).

In order to propagate the faults to the correct outputs and observe their effects on the accuracy of approximate circuits, the fault excitation and propagation as well as forward and backward justifications should be aware of the error margin. This can be achieved with a help of a comparator circuitry added at the output of the DUT which compares the output error of the faulty approximate circuit (fault effect) to the specified error tolerance margin in order to decide whether the test pattern results in a violation of the error margin. The

TABLE II. FAULTY AND FAULT FREE APPROXIMATE ADDER TRUTH TABLE AND OUTPUT ERROR COMPARISON

Inputs A_1A_0-B_1B_0	Exact Output	Approximate Output ($\pm\epsilon$)	FS1-SA1 Output ($\pm\epsilon$)	FS2-SA1 (P_1) Output ($\pm\epsilon$)	FS2-SA1 (P_2) Output ($\pm\epsilon$)
00-00(2.0+2.0)	10000 (4.0)	10000 (0.0)	10001 (+0.25)	10010 (+0.5)	10100 (+1.0)
00-01(2.0+2.25)	10001 (4.25)	10001 (0.0)	10001 (0.0)	10011 (+0.5)	10101 (+1.0)
00-10(2.0+2.5)	10010 (4.5)	10010 (0.0)	10011 (+0.25)	10010 (0.0)	10110 (+1.0)
00-11(2.0+2.75)	10011 (4.75)	10011 (0.0)	10011 (0.0)	10011 (0.0)	10111 (+1.0)
01-00(2.25+2.0)	10001 (4.25)	10001 (0.0)	10001 (0.0)	10011 (+0.5)	10101 (+1.0)
01-01(2.25+2.25)	10010 (4.5)	10000 (-0.5)	10001 (-0.25)	10010 (0.0)	10100 (+0.5)
01-10(2.25+2.5)	10011 (4.75)	10011 (0.0)	10011 (0.0)	10011 (0.0)	10111 (+1.0)
01-11(2.25+2.75)	10100 (5.0)	10010 (-0.5)	10011 (-0.25)	10010 (-0.5)	10110 (+0.5)
10-00(2.5+2.0)	10010 (4.5)	10010 (0.0)	10011 (+0.25)	10010 (0.0)	10110 (+1.0)
10-01(2.5+2.25)	10011 (4.75)	10011 (0.0)	10011 (0.0)	10011 (0.0)	10111 (+1.0)
10-10(2.5+2.5)	10100 (5.0)	10100 (0.0)	10101 (+0.25)	10110 (+0.5)	10100 (0.0)
10-11(2.5+2.75)	10101 (5.25)	10101 (0.0)	10101 (0.0)	10111 (+0.5)	10101 (0.0)
11-00(2.75+2.0)	10011 (4.75)	10011 (0.0)	10011 (0.0)	10011 (0.0)	10111 (+1.0)
11-01(2.75+2.25)	10100 (5.0)	10010 (-0.5)	10011 (-0.25)	10010 (-0.5)	10110 (+0.5)
11-10(2.75+2.5)	10101 (5.25)	10101 (0.0)	10101 (0.0)	10111 (+0.5)	10101 (0.0)
11-11(2.75+2.75)	10110 (5.5)	10100 (-0.5)	10101 (-0.25)	10110 (0.0)	10100 (-0.5)

comparison with the error margin is crucial when dealing with both fixed and floating point approximate circuit testing, as the output error of such circuits depends on the propagation of the faults to different bit positions and the respective values of the output bits. This cannot be easily implemented by putting some extra constraints on conventional ATPG tools (e.g., by masking output pins corresponding to lower order bits). Particularly, in the case of floating point approximate circuits, the output bits (of mantissa part) to which the error should be propagated depends on the value of exponent part, which is decided during internal steps of ATPG for backward and forward justifications.

For the above mentioned constraints, we propose a SAT-based ATPG flow for approximate circuits which inherently considers these requirements.

B. SAT based approximable fault identification methodology

In the proposed approach, a gate-level netlist of the circuit under test (DUT) is given and an equivalent testing circuit is generated. As shown in Figure 2(a), the equivalent test circuitry consists of three main sub-components:

1) A faulty approximate submodule (the so called DUT).
2) A fault-free exact submodule, and
3) A comparator to compare the output difference of the faulty and exact circuits with the given error metrics.

In the comparator logic, first a subtractor circuitry is used to determine the absolute output error of the faulty approximate submodule relative to the exact submodule. Then, a comparator is used to compare the output error with the specified error margin and determine whether the fault violates the error margin. If the error margin is violated, then the comparator circuitry generates an error signal to indicate that the injected fault is a non-approximable fault.

In the proposed flow, first a fault is injected to the output of a faulty gate (fault site) in the approximate submodule. Then, the entire test circuitry is converted into a SAT instance which can be easily solved using a SAT solver as shown in Figure 2(a). The SAT instance is only satisfiable, iff there is at least one input assignment which results in a violation of the error tolerance margin in the presence of a fault (see Figure 2(b)). The SAT solver inherently evaluates all propagation paths from the fault site to any of the primary outputs before it decides whether the fault is approximable or non-approximable. Finally, the tool generates set of test patterns that are capable of detecting the non-approximable faults. This way, we are able to correctly identify and classify approximable faults.

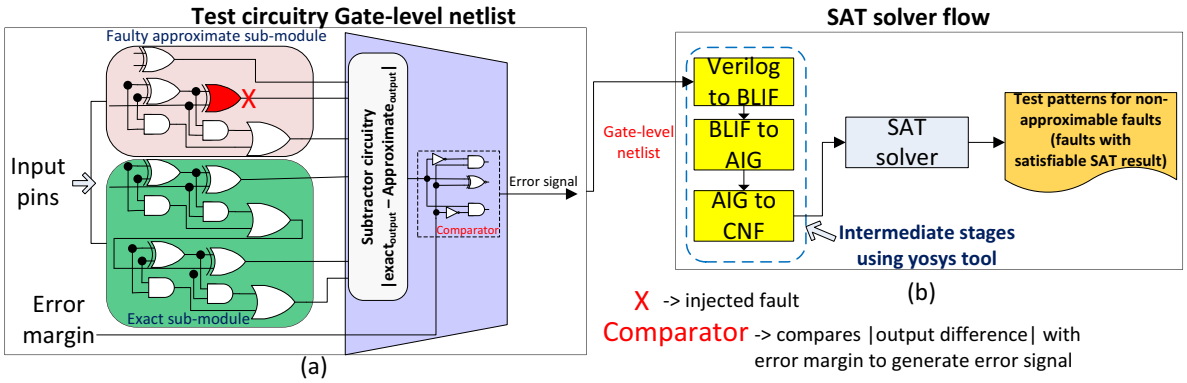


Fig. 2. SAT-based ATPG for approximate circuits (a) faulty test circuitry gate-level netlist generation (b) SAT instance generation and satisfiability testing

C. Overall implementation

The main part of the SAT based approximate circuit testing is implemented in C^{++} as shown in Algorithm 1. The algorithm takes the gate-level netlist of the equivalent test circuitry (exact version, approximate version (DUT) and comparator sub-modules), a fault list and the error tolerance margin as inputs and generates the test patterns and fault coverage as its outputs. The algorithm first prepares a faulty test circuitry by injecting a fault to a gate output (fault site) of the approximate sub-module (Step-4). Then, since the employed SAT solver requires Conjunctive Normal Form (CNF) input, the gate level netlist of entire test circuitry (with injected fault) is converted into a CNF SAT instance through intermediate Berkeley Logic Interchange Format (BLIF) and And-Inverter Graph (AIG) steps using Yosys tool [17] (Step-5). Once the CNF SAT instance is prepared, a SAT solver (RelSAT [18] in our case) is employed to solve the satisfiability of the SAT instance (Step-6). At this stage, it is worth to mention that the comparator circuitry serves as an enforcement tool for the SAT solver to comprehensively check all fault propagation paths in order to determine the satisfiability. Finally, if the SAT outcome is satisfiable, the fault is considered as a non-approximable and the SAT input assignments are used as the test patterns to detect it (Steps 7-9). Otherwise, the fault is approximable and can be ignored safely (Step-11). This process is iteratively applied to all faults available in the fault list. The inclusion of the error metric in the CNF SAT instance helps to avoid over-testing as the test patterns that detect the non-approximable faults are the only test vector needed for post manufacturing testing.

Algorithm 1: Boolean satisfiability based approximate circuit test pattern generation

```
1  function: Fault identification SAT flow;
   Input  : Testing circuit gate-level netlist, fault list (F_l)
   Output: test-patterns, fault coverage
2  Test_pattern ← NULL;
3  foreach f ∈ F_l do
4      faulty_test circuitry ← inject(approximate_module, f);
5      CNF ← Yosys(faulty_test circuitry);
6      SAT_outcome ← SAT_solver(CNF);
7      if SAT_outcome == satisfiable then
8          Non-approximable_faults_i ← f;
9          Test_pattern ← Satisfiable input assignment;
10         i ← i+1;
11     else
12         approximable_faults ← f;
13     end
14 end
15 return Test_pattern and fault coverage;
```

The flow presented in Figure 2 is generic and can be applied to any approximate circuit by configuring the comparator submodule. For instance, a proper fault excitation and propagation to the right outputs of floating point approximate circuits can be achieved by configuring the comparator to calculate the errors of the mantissa and exponent fields separately. This is necessary because the error margin of floating point approximate circuits depends on both mantissa and exponent values, and not just only which output bits the fault is propagated. Thus, the comparator first determines the errors in the mantissa and exponent parts independently. Afterwards, the overall error (the sum of mantissa and exponent errors) is compared to the tolerance margin and the error signal is generated if the margin is violated. This enables the propagation of faults to either fields and determines their approximability based on the violation of the tolerance margin.

Similarly, the comparator circuit for fixed-point approximate circuits is configured to handle the integer and fractional parts separately. Unlike, floating point circuits, the output error of fixed-point approximate circuits highly depends on the position of the binary point which determines the bit width of integer and fractional parts as well as the weights of their bit positions. Therefore, the output error comparator should consider the binary point position when calculating the output errors of the integer and fractional parts and compare them to the specified tolerance margin. Hence, the faults in fixed-point approximate circuits should be propagated either to the integer or fractional parts to violate the tolerance margin.

IV. Experimental Results

A. Floating point circuit evaluation

Floating-Point Units (FPUs) have higher computational capability and flexibility than integer units, however, they are more complex and power consuming circuits. As a result several approximate techniques have been proposed to improve their complexity and power consumption. Therefore, it is crucial to target approximate FPU circuits for approximate testing and demonstrate the potentials of error tolerance of approximate FPUs. Hence, we evaluated the test aspect of basic single precision IEEE-754 standard floating point circuits.

To maximize approximation benefit of FPU, the lower order bits of both mantissa and exponent part of the FPU circuits are subjected to inexact computation for a given error tolerance margin. To achieve this, we prepared approximate FPU circuits by replacing the exact adder and multiplier

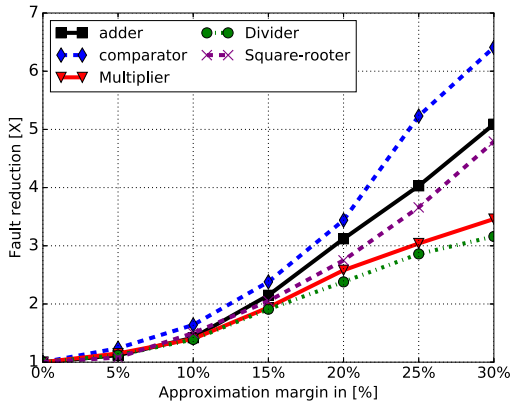


Fig. 3. Approximation induced fault reduction of approximate floating point circuits for different levels of approximation (error tolerance) margin

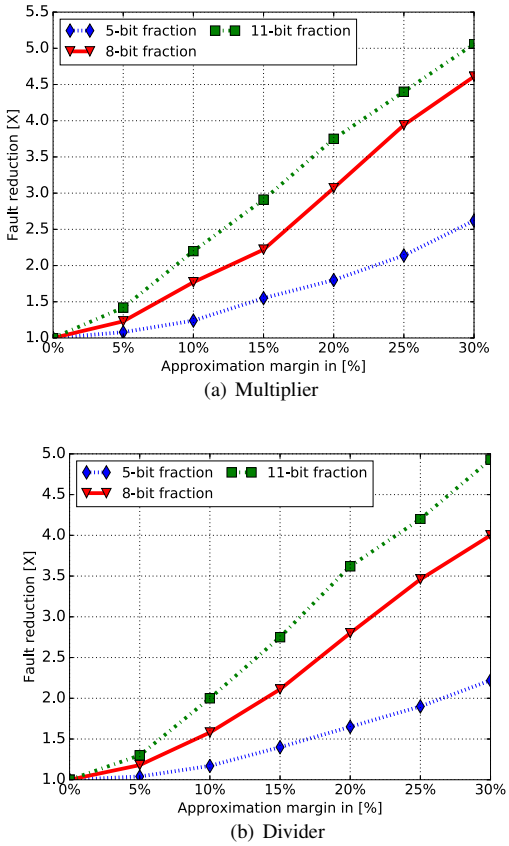
units of the FPU circuits by their approximate variants [6]. Afterwards, our approximate testing flow is applied to test the approximate FPU circuits. The approximation benefit of FPU can be improved further by changing the bit width of the mantissa and exponent parts as it determine the precision and range of the representable values. The evaluation is performed for single precision IEEE-754 standard (1-bit for sign, 23-bits mantissa and 8-bits exponent) as it is the most commonly used representation. However, flexible mantissa bit width is evaluated using fixed-point circuits as presented in Section IV-B.

The fault detection and reduction potential of the proposed approach is evaluated for different error tolerance (approximation) margins, as shown in Figure 3. The figure shows the fault reduction obtained using the proposed approach for wide approximation margin. When the approximation margin is less, more faults are categorized as non-approximable and hence, must be detected. For example, for 0% margin the proposed flow has to detect all possible faults and hence, it will serve as a conventional ATPG flow. However, more and more faults can be ignored (approximated) with increase in the approximation (error tolerance) margin, which eventually leads to significant improvement in the fault reduction capability of our approach. Since the output error rate increases with increase in the approximation margin, a trade-off can be made between output error rate and test cost reduction in order to decide the approximation margin.

B. Fixed-point circuit evaluation

In several signal processing applications, fixed-point operation is used as a simpler and less complex alternative to deal with real numbers as it does not require a dedicated hardware. Unlike IEEE-754 standard FPUs, the bit width of the decimal (mantissa) and the fraction part of a fixed-point number can be changed. This allows us to make design decision between precision and representation range of fixed-point numbers. Hence, the relaxation in bit width can be exploited in order to add another approximation level in the design and testing of approximation circuits.

For this purpose, we used 32-bits fixed-point approximate multiplier and divider circuits to evaluate the fault reduction potential of the proposed technique for 5, 8 and 11 bits fraction parts (i.e., varying precision) as shown in Figure 4. Figure 4(a) shows the approximate 32-bit multiplier fault reduction using 5, 8 and 11 bits fraction parts. It can be easily observed that



(a) Multiplier

(b) Divider

Fig. 4. Fault reduction of approximate fixed-point circuits for different levels of approximation margin and varying fractional bit widths

the usage of more fraction bits leads to less observable (non-approximable) faults and hence, better fault reduction. This is mainly because more fraction bit width has better precision and higher approximation potential. Similar pattern is observed for the 32-bit fixed-point divider as shown in Figure 4(b). Therefore, for both circuits the non-approximable faults can be reduced by more than 2× when the fraction bit-width is increased from 5 to 11 bits.

C. Comparison with related work

In order to demonstrate the advantages of the proposed technique, it is compared with a state-of-the-art SAT based augmented ATPG testing technique presented in [9]. The work in [9] uses SAT to pre-process the faults before applying conventional ATPG tool to generate the test patterns for the approximate circuit. Hence, in their work, both exact and faulty approximate designs are provided to the ATPG tool to determine whether a fault is approximable or not by comparing the ATPG output difference of the two circuits with the specified error margin. Such an approach is vulnerable to false-approximable and false-undetectable faults as the conventional ATPG tool is inherently unaware of approximation constraints.

Table III presents the fault detection and coverage comparison of baseline (approximation unaware), augmented ATPG [9] and the proposed approximation-aware SAT-based ATPG methodologies for different approximate FPU circuits using 20% error tolerance margin. In the table, the first three columns specify the circuit type, number of gates and number of injected faults, respectively. The fault coverage and detected faults of the baseline test generation (i.e., conventional test flow

TABLE III. FAULT CLASSIFICATION, DETECTION AND COVERAGE COMPARISON OF THE PROPOSED APPROXIMATION AWARE SAT BASED ATPG AND AUGMENTED ATPG [9] FOR DIFFERENT APPROXIMATE FPU CIRCUITS WITH 20% ERROR TOLERANCE MARGIN

FPU circuits	#Gates	#Faults injected	Baseline detected (coverage in %)	Related work (augmented ATPG [9])				Proposed approximation aware SAT based ATPG		
				#Non-approximable faults detected (coverage in %)	Fault reduction [X]	False approximable (FA)	False undetected (FUND)	#Non-approximable faults detected (coverage in %)	Fault reduction [X]	SAT-flow runtime (s)
Adder	1247	3494	3396 (97.19%)	1175 (100%)	2.89×	0%	0%	1090 (100%)	3.12×	1055
Comparator	181	1362	1359 (99.77%)	501(100%)	2.71×	0%	0%	395 (100%)	3.44×	20
Multiplier	2009	5018	4817 (95.9%)	2032 (98.2%)	2.37×	1%	0.3%	1864 (99.5%)	2.58×	1796
Divider	2230	6460	6162 (95.38%)	2810 (97.69%)	2.19×	1.1%	0.21%	2588 (99%)	2.38×	2688
Sqrt	14066	20000	18240 (91.2%)	8723 (93.5%)	2.09×	2%	0.5%	7273 (96%)	2.5×	16321
Average			**Coverage= 95.8%**	**Coverage= 97.8%**	**2.45×**	**0.82%**	**0.2%**	**Coverage= 98.9%**	**2.85×**	**4376**

without consideration of approximation) is given in the fourth column. The fault coverage, reduction and misclassification of the related work [9] are presented in the fifth to eighth columns followed by the proposed approximation-aware SAT-based ATPG results. As it can be seen from the table, the proposed technique has an average of 2.85× reduction in the detected faults (non-approximable faults) while maintaining high fault coverage. The table also shows that our approach has better fault reduction and coverage than the work in [9].

The table also presents the percentage of false approximable (FA) and false undetectable (FUND) faults of the method in [9]. FA are non-approximable faults (faults resulting in violation of the error margin) which are wrongly classified as approximable while FUND are detectable faults wrongly classified as undetectable. These two misclassifications are critical and can severely affect the fault coverage and test quality and hence, they should be detected in order to improve the fault coverage. As can be seen in the table, the work in [9] results in various misclassifications while the proposed technique resolves this problem.

D. SAT optimization and test pattern compaction

Although SAT based ATPG has potential improvement in generating comprehensive and high coverage test patterns than conventional ATPG, it has two main limitations that needed to be addressed [19]. The first downside of SAT based ATPG is that it takes relatively longer runtime then the conventional ATPG approach [19]. This challenge is more important when dealing with larger and complex circuits. However, the runtime of SAT based ATPGs can be accelerated by using various optimization techniques such as powerful SAT solving algorithms [20] and parallel execution approaches. Additionally, the runtime of SAT based ATPG can be improved by reducing and optimizing the CNF representation of circuits [21]. CNF optimization [21] is also important in improving the scalability of SAT based ATPG to deal with larger and more complex circuits. Since these techniques are orthogonal to our approach they can be easily applied to improve the runtime of our SAT-based approximate ATPG framework.

The second drawback is that SAT-based ATPG algorithms have less powerful test pattern compaction ability than the conventional ATPG algorithms [19]. However, several techniques have been proposed to improve the test compaction abilities of SAT-based ATPG [19, 22, 23]. These test pattern compaction techniques can be combined with the proposed SAT based test pattern generation to improve its compaction capabilities.

V. CONCLUSIONS

As approximate computing is becoming a popular computing paradigm, it has become imperative to have a testing framework capable of exciting and propagating faults while being aware of the approximation characteristics of these circuits under test. In this paper, we presented a boolean satisfiability based approximate test pattern generation technique to address the test challenges of approximate circuits. For a given approximate circuit, a SAT solver is employed to identify the non-approximable faults and generate the test patterns with high fault coverage. Experimental results show that, the proposed approach can reduce the pattern count and test time by 2.85× on average while maintaining high fault coverage. Moreover, the proposed approach can effectively detect all false-approximable and false-undetectable faults of approximate circuits, compared to state of the art techniques.

REFERENCES

[1] T. Moreau *et al.*, "Snnap: Approximate computing on programmable socs via neural acceleration," in *HPCA*, 2015.
[2] Q. Zhang *et al.*, "Approxit: An approximate computing framework for iterative methods," in *DAC*, 2014.
[3] A. Gebregiorgis *et al.*, "Error propagation aware timing relaxation for approximate near threshold computing," in *DAC*, 2017.
[4] Z. Yang *et al.*, "Approximate xor/xnor-based adders for inexact computing," in *Nanotechnology (IEEE-NANo)*, 2013.
[5] J. Huang *et al.*, "A methodology for energy-quality tradeoff using imprecise hardware," in *DAC*, 2012.
[6] A. Yazdanbakhsh *et al.*, "Axbench: A multiplatform benchmark suite for approximate computing," *IEEE Design & Test*, 2017.
[7] S. Venkataramani *et al.*, "Salsa: systematic logic synthesis of approximate circuits," in *DAC*, 2012.
[8] M. Traiola *et al.*, "Testing approximate digital circuits: Challenges and opportunities," in *LAmTS*, 2018.
[9] A. Chandrasekharan *et al.*, "Approximation-aware testing for approximate circuits," in *ASP-DAC*, 2018.
[10] L. Anghel *et al.*, "Test and reliability in approximate computing," *Journal of Electronic Testing*, 2018.
[11] M. Traiola *et al.*, "On the comparison of different atpg approaches for approximate integrated circuits," in *DDECS*, 2018.
[12] I. Wali *et al.*, "Towards approximation during test of integrated circuits," in *DDECS*, 2017.
[13] I. Wali *et al.*, "Can we approximate the test of integrated circuits?" in *Workshop on Approximate Computing*, 2017.
[14] S. Eggersgluess *et al.*, "A highly fault-efficient sat-based atpg flow," *IEEE Design & Test of Computers*, 2012.
[15] M. Sauer *et al.*, "Efficient sat-based search for longest sensitisable paths," in *Asian Test Symposium (ATS)*, 2011.
[16] S. Anwar *et al.*, "Fixed point optimization of deep convolutional neural networks for object recognition," in *ICASSP*, 2015.
[17] C. Wolf, "Yosys open synthesis suite," 2016.
[18] "Relsat 2.1," www.bayardo.org/resources.html.
[19] S. Eggersglüß *et al.*, "Improved sat-based atpg: More constraints, better compaction," in *ICCAD*, 2013.
[20] B. Becker *et al.*, "Recent advances in sat-based atpg: Non-standard fault models, multi constraints and optimization." in *DTIS*, 2014.
[21] J. Balcarek *et al.*, "Techniques for sat-based constrained test pattern generation," *Microprocessors and Microsystems*, 2013.
[22] S. Eggersglüß *et al.*, "A new sat-based atpg for generating highly compacted test sets," in *DDECS*, 2012.
[23] M. Sauer *et al.*, "Efficient sat-based dynamic compaction and relaxation for longest sensitizable paths," in *DATE*, 2013.