Approximation-aware Testing for Approximate Circuits*

Arun Chandrasekharan[1] Stephan Eggersglüß[1,2] Daniel Große[1,2] Rolf Drechsler[1,2]
[1]Group of Computer Architecture, University of Bremen, Germany
[2]Cyber Physical Systems, DFKI GmbH, Bremen, Germany
{arun,segg,grosse,drechsle}@cs.uni-bremen.de

***Abstract*—A wide range of applications significantly benefit from the Approximate Computing (AC) paradigm in terms of speed or power reduction. AC achieves this by tolerating errors in the design. These errors are introduced into the design either manually by the designer or by approximate synthesis approaches. From here, the standard design flow is taken. Hence, the manufactured AC chip is eventually tested for production errors using well established fault models. To be precise, if the test for a test pattern fails, the AC chip is sorted out. However, from a general perspective this procedure results in throwing away chips which are perfectly fine taking into account that the considered fault (i.e. physical defect that leads to the error) can still be tolerated because of approximation. This can lead to a significant amount of yield loss.**

In this paper, we present an approximation-aware test methodology which can be easily integrated into the regular test flow. It is based on a pre-process to identify approximation-redundant faults. By this, we remove all potential faults that no longer need to be tested because they can be tolerated under the given error metric. Our experimental results and case studies on a wide variety of benchmark circuits show a significant potential for yield improvement.

I. Introduction

The complexity of chips is steadily increasing while, at the same time, the feature sizes are shrinking. Both facts pose major challenges to the design of todays chips. In addition, better energy efficiency and performance become a major concern. A promising solution is the emerging *Approximate Computing* (AC) design paradigm. The key idea of AC is to trade off correct computation against energy or performance. The good news is that there are many crucial resource-hungry applications (e.g. audio/video processing, learning, big-data analysis) which can tolerate some deviation of the exact result [1]. From the hardware design side, various handcrafted approximate designs have been proposed, ranging from building blocks such as adders, multipliers, etc. to complex configurable CPU architectures [2]. In the context of design automation for AC, also solutions for synthesis [3], verification [4], simulation [5] etc., have been proposed. In this paper, we focus on the last design step of the chip design, i.e. the post production test. In general, the task of manufacturing test is to detect whether a physical defect is present in the chip or not. If yes, the chip will not be shipped to the customer. However, given an approximate circuit and a physical defect, the crucial question is, whether the chip still can be shipped since the defect can be tolerated under approximation. If we can provide a positive answer to this question, this leads to a significant potential for yield improvement.

In this paper we present an approximation-aware test methodology. To the best of our knowledge this is the first approach considering *the impact of design level approximations in post production test*. Our methodology does not radically change the test flow, rather it is a pre-process to classical *Automatic Test Pattern Generation* (ATPG). The key idea is to classify each fault (the logical manifestation of a defect) in *approximation-redundant* or *non-approximation*. For this task, we essentially compare the non-approximated (golden) design against the approximated design with an injected fault under the considered error metric constraint. Using formal methods (SAT and variants) as well as structural techniques allow us to classify the fault. For a wide range of benchmarks, we demonstrate the advantage of our approach. We show that, depending on the concrete approximation and error metric (which is driven by the application), a relative reduction of up to 80% in fault count can be achieved.

In summary, this paper makes the following contributions:

- Identification of approximation-redundant faults
- Mapping into a formal fault classification problem
- Pre-process to classical ATPG
- Significant yield improvement potential

II. Related Work

Several works have been proposed to improve the yield by classifying faults as *acceptable* faults and *unacceptable* faults. These employ different techniques such as integer linear programming [6], sampling methods for error estimation [7], threshold based test generation [8] etc. Further, the work [9] shows a technique to generate tests efficiently if such a classification is available.

However, all these approaches are applied to conventional circuits without taking into consideration the errors introduced as part of the design process itself. Therefore, these approaches cannot be directly applied to AC. It has to be noted that "normal" circuits that produce errors due to manufacturing defects do not constitute approximation circuits. In AC, errors are introduced into the design for high speed or low power. In other words the error is already introduced and taken into consideration during design time. Now if for these designed approximated circuits arbitrary fabrication errors are allowed, the error effects will magnify. For instance, if we discard all the stuck-at faults at the lower bit of an approximation adder under a worst-case error constraint of at most 2, the resulting error can in fact increase above the designed limit. Therefore, the AC application will fail under such defects. This is exemplified later in a motivating example in Section IV-A. The key of our

* This work was supported by the German Research Foundation (DFG) within the project MANIAC (DR 287/29-1), the Reinhart Koselleck project DR 287/23-1, the University of Bremen's graduate school SyDe, funded by the German Excellence Initiative, and by the German Academic Exchange Service (DAAD).

work is that we identify all faults which are guaranteed not to violate the given error metric constraint, coming from the AC application. This ensures that the AC chip will work as originally envisioned for.

At this point we differentiate our work from [10]. In [10] (and closely related [11]), structural analysis is used to determine the most vulnerable circuit elements. Only for those elements test patterns are generated and this approach is called *approximate test*. In addition, note that [10] targets "regular" non-approximated circuits and therefore we categorize it as a technique for *approximating a test*, rather than a technique for testing an already approximated circuit.

III. Preliminaries

At first, the relevant parts from post production tests are reviewed in this section. Then, the basic error metrics typically used in approximate computing are defined and it is reviewed how to precisely compute them.

A. Post Production Test, Faults and ATPG

The manufacturing process of a circuit is vulnerable to a large number of physical *defects*, especially due to the shrinking feature sizes. A post production test is applied to the manufactured ICs to detect these defects and filter out the non-correct circuits. A *fault* f is a logical manifestation of these defects. The most popular fault model used in practice is the *Stuck-At Fault Model* (SAFM). In this scheme, a signal connection s in the circuit is considered to be permanently 'stuck' at a constant value, either 1 or 0. In this work, we concentrate only on the SAFM, but the proposed approach can also be extended to other fault models.

A test set T is a set of test vectors $t_1, \ldots, t_n$ applied at the circuit inputs which activates the fault locations and produces detectable difference at an observation point. In the post-production test, each detectable fault in the circuit has to be covered by at least one test pattern in the test set. The computation of this test set is called *Automated Test Pattern Generation* (ATPG) [12]. The ATPG takes all faults $F = f_1, \ldots, f_m$ of a fault model as input and generates a favorably small test set with a high fault coverage.

Basically, a fault f can be classified by the ATPG in three categories. A fault is called *detectable*, when the ATPG proves that the fault is testable by producing a test which detects f. A fault is *redundant*, when the ATPG proves that there is no test which is able to detect f. A fault is classified as *aborted*, when the ATPG cannot classify f due to reasons of complexity. ATPG techniques are well developed and are able to produce small test sets in reasonable time using structural implication techniques or formal proof engines [13].

B. Error Metrics

Several error metrics have been proposed to determine the quality of approximations (see e.g. [4], [14], [5]). The metrics relevant for this work are given in the following.

Let N be a netlist with p input bits and q output bits, and N_f be the version with an injected fault f. The error e_f of the netlist under the fault f is the absolute difference in the magnitudes of the output. i.e., error e_f over q output bits is

$$e_f = \left|(N_{f,q-1}, ..., N_{f,0}) - (N_{q-1}, ..., N_0)\right| \qquad (1)$$

Here $(N_{f,q-1}, ..., N_{f,0})$ and $(N_{q-1}, ..., N_0)$ are the output bit vectors of the faulty and original netlist respectively.

The *worst-case error* or *error-significance* is the maximum possible error magnitude among all the 2^p input combinations.

$$wc = \max\{e_{f_0}, e_{f_1}, ..., e_{f_M}\} \qquad (2)$$

where $M = 2^p - 1$.

The *error-rate* is the ratio of the *count* of all the errors to the total number of input vectors. i.e.,

$$er = \sum_{i=0}^{M} (e_f \neq 0) \ / \ 2^p \qquad (3)$$

The total bit-flips e_f^h is the hamming distance between N and N_f due to the injected fault f.

$$e_f^h = \sum_{i=0}^{q-1} (N_{f,q-1}, ..., N_{f,0}) \oplus (N_{q-1}, ..., N_0) \qquad (4)$$

And the bit-flip error is the maximum hamming distance possible due to the fault f,

$$bf = \max\{e_{f_0}^h, e_{f_1}^h, ..., e_{f_M}^h\} \qquad (5)$$

All these error metrics are independent quantities on their own and do not necessarily correlate with each other. For instance, a design with very high worst-case error does not imply that the bit-flip error or the error-rate is high.

An approach based on formal methods has been presented in [4] to precisely determine the impact of approximation wrt. a given error metric. The authors propose an approximation miter circuit inspired from the classical formal equivalence checking. In our work we retain the fundamental concepts used in [4], but adapted for fault classification.

IV. Approximation-aware Test Methodology

In this section we introduce the proposed approximation-aware test methodology. Before the details are provided, we describe the general idea using a motivating example. In the second half we present the proposed fault classification approach.

A. General Idea and Motivating Example

In the context of approximate computing yield improvement can be achieved when a fault (logical manifestation of a physical defect) is found which can still be tolerated under the given error metric. In this case the fabricated chip can still be used as originally intended, instead of sorting it out. As mentioned earlier we consider single stuck-at faults in this work only (cf. Section III-A). Given an approximate circuit, a constraint wrt. an error metric, the list of all faults for the approximate circuit, then each fault is categorized by our approach into one of the following:

- *approximation-redundant fault* – These are faults which can be approximated, i.e. the fault effect can have an observable effect on the outputs, but it is proven that the effect will always be below the given error limit. Hence, no test pattern is needed for these faults. Note that regular redundant faults are also classified into this category.
- *non-approximation fault* – These are faults whose error behavior is above the given error limit. Hence, they have

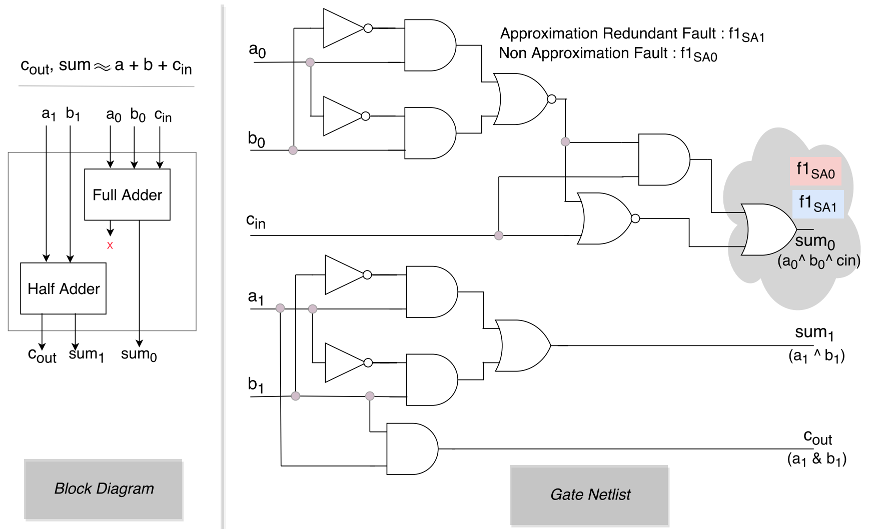

Fig. 1. Approximation adder with faults and truth table

	Correct†	Appx‡		Appx:SA0*		Appx:SA1$^\pm$	
In	Out†	Out‡	$e^\ddagger$	Out*	$e^\star$	Out$^\pm$	$e^\pm$
00000	000	000	0	000	0	001	1
00001	001	001	0	000	1	001	0
00010	010	010	0	010	0	011	1
00011	011	011	0	010	1	011	0
00100	001	001	0	000	1	001	0
00101	010	000	2	000	2	001	1
00110	011	011	0	010	1	011	0
00111	100	010	2	010	2	011	1
01000	010	010	0	010	0	011	1
01001	011	011	0	010	1	011	0
01010	100	100	0	100	0	101	1
01011	101	101	0	100	1	101	0
01100	011	011	0	010	1	011	0
01101	100	010	2	010	2	011	1
01110	101	101	0	100	1	101	0
01111	110	100	2	100	2	101	1
10000	001	001	0	000	1	001	0
10001	010	000	2	000	2	001	1
10010	011	011	0	010	1	011	0
10011	100	010	2	010	2	011	1
10100	010	000	2	000	2	001	1
10101	011	001	2	000	3	001	2
10110	100	010	2	010	2	011	1
10111	101	011	2	010	3	011	2
11000	011	011	0	010	1	011	0
11001	100	010	2	010	2	011	1
11010	101	101	0	100	1	101	0
11011	110	100	2	100	2	101	1
11100	100	010	2	010	2	011	1
11101	101	011	2	010	2	011	2
11110	110	100	2	100	2	101	1
11111	111	101	2	100	3	101	2

†, ‡ Golden non-approx, approx (carry cut) 2-bit adder responses
⋆, ± Approx adder with SA0, SA1 at sum_0 ($\mathrm{f1}_{\mathrm{SA0}}$, $\mathrm{f1}_{\mathrm{SA1}}$)
In:$\mathrm{C}_{\mathrm{in}}\mathrm{a}_1\mathrm{a}_0\mathrm{b}_1\mathrm{b}_0$, Out:$\mathrm{C}_{\mathrm{out}}\mathrm{sum}_1\mathrm{sum}_0$
e: error in each case, worst-case errors $\mathrm{wc}^\ddagger$=2,$\mathrm{wc}^\star$=3, $\mathrm{wc}^\pm$=2

to be tested in the post production test and thus a test pattern has to be generated for these faults.

Further, if a fault cannot be classified due to reasons of excessive run time, it is treated as an non-approximation fault.

In the following a motivating example is provided to demonstrate both fault categories. Consider the 2-bit approximation adder as shown in Fig. 1. This adder has two 2-bit inputs $a = a_1a_0$ and $b = b_1b_0$ and the carry input c_{in} and computes the sum as $c_{out}sum_1sum_0$. The (functional) approximation has been performed by cutting the carry from the full adder to the half adder as can be seen in the block diagram on the left of Fig. 1. As error metric we consider a worst-case error of 2 (coming from the application where the adder is used). To explain the proposed fault classification we will focus on the output bit sum_0 and the faults at this bit, i.e. $\mathrm{f1}_{\mathrm{SA0}}$ and $\mathrm{f1}_{\mathrm{SA1}}$ corresponding to a stuck-at-0 and stuck-at-1 fault, respectively.

The truth table of the original golden adder, the approximation adder, and the approximation adder with different fault manifestations is given at the right side of Fig. 1. The first column of the truth table is the input applied during fault simulation, followed by the output response of the correct golden adder. Next the response of the approximation adder, and the error $e^\ddagger$ (as an integer) is shown. The worst-case error $\mathrm{wc}^\ddagger$ is the maximum among all such $e^\ddagger$. As can be seen the maximum is 2, since cutting the carry leads sometimes to a "wrong" computation but the deviation from the correct result is always less than or equal to 2. The next four columns are the output and error response of the approximation adder with the stuck-at fault, i.e. SA0 and SA1 at the sum_0 output bit. Recall, since the adder is used for AC applications all the errors below the worst-case error of $\mathrm{wc}^\ddagger = 2$ are tolerated. Under this error criteria, the SA1 fault $f1_{SA1}$ at the sum_0 output bit is approximation-redundant because error $e^\pm$ is always less than or equal to 2, as can be seen in the rightmost column of the truth table. However, for the same output bit, the SA0 fault $f1_{SA0}$ is a non-approximation fault: the worst-case error is 3 which becomes evident in column $e^\star$ and the shaded rows.

Algorithm 1 Approximation-aware fault classification

function APPROX_PREPROCESS(faultList $faults$, error behav. e)
  N ← get_network()
  for each $f \in faults$ **do**
    if fault_not_processed(f) **then**
      N_f ← get_faulty_network(f)
      E ← get_error_computation_netw(metric(e))
      C ← negation_of(e)
      Φ = construct_miter(N, N_f, E, C)
      result = solve(Φ)
      if result = SAT **then**
        set $f_{status} \leftarrow NonApproxFault$
      else
        set $f_{status} \leftarrow ApproxFault$
      end if
      imply_approximation (f, f_{status})
    end if
  end for
  return *faults*
end function

In practice, the employed error criteria follows the requirements of the AC application. Each application will have a different sensitivity on the error metric given in Section III-B. However, if we can identify many approximation-redundant faults, they do not have to be tested since they can be tolerated based on the given error metric constraint.

In the next section we present the proposed fault classification algorithm which can handle the different error metrics.

B. Approximation-aware Fault Classification

At first, the overall algorithm is presented. Then, the core of the algorithm is detailed.

1) Overall Algorithm: The main part of the proposed approximation-aware fault classification methodology is the fault-preprocessor. It classifies each fault into the above introduced fault categories and is inspired by regular SAT-based ATPG approaches, since these approaches are known to be very effective in proving redundant faults.

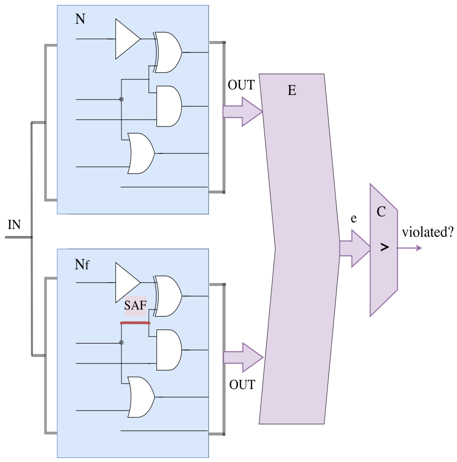

Fig. 2. Approximation Miter for Approximation-aware Fault Classification

The approximation-aware fault classification algorithm is outlined in Algorithm 1. The algorithm is generic and details on the individual steps are given below. The inputs are the list of all faults and the error behavior (in terms of a constraint wrt. an error metric, e.g. the worst case error should be less than 10). Such information can be easily provided by the designer of the approximation circuit. Further, an approximation-aware fault classification miter is constructed (see Line 8). This formulation is then transformed into a SAT instance which is solved by a SAT solver. The general principle of an approximation miter has already been presented in [4] where error metrics are precisely computed. In this work however, we follow the miter principle but use it to determine the fault classification. After fault classification, structural techniques are applied to deduce further faults. The pre-processor algorithm returns the same list of faults, but for each fault the status has been updated, i.e. it has been classified as approximation-redundant or non-approximation. In the following we explain how the approximation miter for fault classification is constructed and used in our approach.

2) Approximation Miter for Fault Classification: The approximation miter for fault classification (see Fig. 2 and Line 8 in Algorithm 1) is constructed using

- the *golden reference netlist* N – this netlist consists of the correct (non-approximated) circuit (provided by get_network() in Line 2)
- the *faulty approximate netlist* N_f – this netlist is the final approximate netlist including fault f (provided by get_faulty_network(f) in Line 5)
- the *error computation network* E – based on the given error metric, this network is used to compute the concrete error of a given output assignment of both netlists (see Line 6)
- the *fault classification network* C – the result of the fault classification network becomes 1, if the comparison of both netlists violates the error metric constraint

Again, the goal of the fault classification miter is to decide whether the current fault is approximation-redundant or not. In other words we are looking for an input assignment such that the given error metric constraint is violated. For instance, in case of the motivating example this worst-case constraint is $\mathrm{wc} \leq 2$, so we are looking for its negation. For this approximate adder example we set C to $\mathrm{wc} > 2$ (see Line 7).

Now the complete problem is encoded as a SAT instance and run a SAT solver. If the solver returns satisfiable – so there is at least one input assignment which violates the error metric constraint – we have proven that the fault is a non-approximation fault (Line 11). If the solver returns unsatisfiable, we have proven that the fault is an approximation-redundant fault (Line 13). This fault does not have to be targeted during the regular ATPG stage.

In addition to the SAT techniques mentioned above, several structural techniques are also used in conjunction with the SAT solver for efficiency (see Line 15). This includes for example fault equivalence rules and constant propagation for redundancy removal.

Besides, several trivial approximation-redundant/ non-approximation faults can be identified. Such trivial faults are located near the outputs. An example is a fault affecting the MSB output bits that always results in error metric constraint violation. These can be directly deduced as non-approximation faults through path tracing.

In the next section the experimental results are provided.

V. Experimental Results

We have implemented all algorithms in C++. The input to our program is the gate level netlist of the approximated circuit which is normally used for standard ATPG generation. Now, instead of running ATPG, we execute our proposed approximation-aware fault classification approach (cf. Section IV). This filters out the approximation-redundant faults. From there on the standard ATPG flow is taken.

In the following we report results for approximated circuits using worst-case error and bit-flip error constraints. Considering error-rate is left out for future work.[1]

The experimental evaluation of our approach has been done for a wide range of circuits. For the circuits we have determined the respective error metrics using the public available tool from [4]. In this section we first explain the results using the worst-case error as approximation pre-processing criteria. These results are provided in Table I. The experimental evaluation using the bit-flip error metric is separately explained at the end of this section with Table II. Note that a combination of these error metrics can also be provided to the tool. All the experiments have been carried out on a system with 3.0GHz Intel Xeon CPU. Further, the worst-case error and the bit-flip error are the error metrics coming from the application.

A. Results for the Worst-Case Error Metric

All the results for the worst-case error scenario are summarized in Table I. Before we describe the different benchmarks (four different sets in total) and the obtained results we explain the general structure of the table. The first three columns give the circuit details such as the number of primary inputs/outputs and the gate count. This is followed by the fault count without our proposed approach, i.e. this gives the "normal" number of faults for which ATPG is executed. Note that fault-equivalence and fault-dominance are already accounted in these fault counts (column: $\mathrm{f}_{\mathrm{orig}}$). The next

[1]As explained before our methodology uses SAT calls to determine the worst-case error behavior (cf. Section IV). However, for error-rate not only pure SAT-calls are needed, but model counting. Model counting is a higher complexity problem compared to SAT (#P-complete vs NP-complete) [23].

TABLE I

SUMMARY OF THE APPROXIMATION-AWARE FAULT CLASSIFICATION RESULTS FOR THE WORST-CASE ERROR

Architecturally approx. adders[1] (set:1)			#Faults			time	EPFL benchmarks[3] (set:3)			#Faults			time
Circuit	#PI/#PO	#gates	f_{orig}	f_{final}^{wc}†	f_{Δ}^{wc}(%)	sec	Circuit	#PI/#PO	#gates	f_{orig}	f_{final}^{wc}†	f_{Δ}^{wc}(%)	sec
ACA_II_N16_Q4 ±	32/17	225	483	180	62.73%	14s	Barrel shifter*	135/128	3975	8540	6677	21.81%	3493s
ACA_II_N16_Q8	32/17	255	535	277	48.22%	16s	Max*	512/130	3780	7468	5783	22.56%	2156s
ACA_I_N16_Q4	32/17	256	530	174	67.17%	14s	Alu control unit*	7/26	178	378	252	33.33%	5s
ETAII_N16_Q8 ∓	32/17	255	535	277	48.22%	16s	Coding-cavlc*	10/11	885	1830	1194	34.75%	73s
ETAII_N16_Q4	32/17	225	483	180	62.73%	13s	Lookahead XY router*	60/30	370	739	459	62.11%	12s
GDA_St_N16_M4_P4‡	32/17	258	575	331	42.43%	17s	Adder*	256/129	1644	3910	2738	29.97%	969s
GDA_St_N16_M4_P8	32/17	280	617	188	69.53%	21s	Priority encoder*	128/8	1225	2759	1335	51.61%	84s
GeAr_N16_R2_P4‡‡	32/17	255	541	160	70.43%	16s	Decoder*	8/256	571	2338	2175	6.97%	132s
GeAr_N16_R6_P4	32/17	263	561	286	49.02%	19s	Round robin*	256/129	16587	26249	11802	55.04%	43940s
GeAr_N16_R4_P8	32/17	261	552	161	70.83%	17s	Sin*	24/25	5492	13979	12756	8.74%	7464s
GeAr_N16_R4_P4	32/17	255	535	277	48.22%	16s							
Arithmetic designs[2] (set:2)			#Faults			time	ISCAS-85 benchmarks[4] (set:4)			#Faults			time
Circuit	#PI/#PO	#gates	f_{orig}	f_{final}^{wc}†	f_{Δ}^{wc}(%)	sec	Circuit	#PI/#PO	#gates	f_{orig}	f_{final}^{wc}†	f_{Δ}^{wc}(%)	sec
Han Carlson Adder*	64/33	655	1415	969	31.52%	88s	c499*	41/32	577	1320	755	42.80%	53s
Kogge Stone Adder*	64/33	839	1789	1475	17.55%	140s	c880*	60/26	527	1074	271	74.77%	27s
Brent Kung Adder*	64/33	545	1178	700	40.58%	51s	c432*	36/7	256	487	441	09.45%	7s
Wallace Multiplier*	16/16	641	1641	669	59.23%	5027s	c1355*	41/32	575	1330	680	48.87%	57s
Array Multiplier*	16/16	610	1585	619	60.95%	4250s	c1908*	33/25	427	974	694	28.74%	46s
Dadda Multiplier*	16/16	641	1641	652	59.40%	6875s	c2670*	233/140	931	1950	372	80.92%	138s
MAC unit1*	24/16	725	1821	760	58.26%	12782s	c3540*	50/22	1192	2657	2388	10.12%	268s
MAC unit2*	33/48	874	2104	492	76.61%	921s	c5315*	178/123	2063	4224	2851	32.50%	1112s
4-Operand Adder*	64/18	614	1434	1156	19.39%	60s	c7552*	207/108	2013	4490	2938	34.57%	1014s

#PI, #PO: number of primary inputs, primary outputs. #gates: gate count after synthesis *time*: time taken for f_{final}^{wc}

f_{orig} : final fault count for which ATPG generated without approximation (dominant, equivalent faults not included)

f_{final}^{wc}: final fault count after approx. pre-processor with worst-case limits. f_{Δ}^{wc}: relative reduction in fault(%). $f_{\Delta}^{wc} = \left(f_{orig} - f_{final}^{wc} \;/\; f_{orig}\right) * 100$

$*$ shows approximation using public tool [15], further taken through standard synthesis flow. † worst-case error evaluated using [4]

[1] Adhoc architecturally approx adders: ±ACA [16], ∓ETA [17] ‡GDA [18], ‡‡GeAr [19]. [2] Arith from [20]. [3] from [21]. [4] from [22]

two columns provide the resulting fault count and reduction in faults using our approximation-aware fault classification methodology (columns: f_{final}^{wc} and f_{Δ}^{wc} %). The last column denotes the run-time in CPU seconds spent for our developed approach, i.e. only the pre-processing (Algorithm 1).

1) Arithmetic Circuits: The first two sets in Table I consists of commonly used approximation arithmetic circuits. The first set are manually architected approximation adders primarily used in image processing applications [16], [17], [18]. These designs are available in the repository [19]. As evident from the Table I, a significant portion of the faults in all these designs are approximation-redundant. It can also be seen that such architectural schemes show a wide range in approximation-redundant fault count, even in the same category. For example among the different *Almost Correct Adders* [16], ACA_I_N16_Q4 has a far higher ratio of approximation faults compared to the scheme ACA_II_N16_Q8 (67% vs 48%). The adder GDA_St_N16_M4_P4 [18] has the least ratio of approximation faults in this category, about 42%.

In the second set, other arithmetic circuits such as fast adders, multipliers, multiply accumulate (MAC) etc., are evaluated. These designs are from [20]. The automated approximation scheme (public available on GitHub [15]) has been used to approximate these circuits. Similar to the architecturally approximated designs, the relative mix of approximation-redundant and non-approximation faults in these circuits also vary widely depending on the circuit structure.

2) Other Standard Benchmark Circuits: We also have evaluated our approach on circuits from the ISCAS-85 [22] and EPFL [21] benchmarks to demonstrate its generality. Our methodology is able to classify a high percentage of faults as approximation-redundant to be skipped from ATPG generation, eventually improving the yield. The highest fraction of approximation-redundant faults is obtained in the ISCAS-85 circuit C2670 (above 80%). However, there is a wide variation in the relative percentage of faults classified as approximation-redundant. This primarily stems from the structure of the circuit, approximation scheme employed and the error tolerance of the AC application.

B. Results for the Bit-Flip Error Metric

We have taken the same set of designs given in Table I for the evaluation of the approximation-aware fault classification methodology under the bit-flip error metric. The results obtained are summarized in Table II. As mentioned before the bit-flip error is the maximum hamming distance of the output bits of the approximated and non-approximated designs, irrespective of the error magnitude. The Table II shows the approximation-aware fault classification results for architecturally approximated adders [16], [17], [18], [19], arithmetic designs [20], standard ISCAS benchmark circuits [22] and EPFL benchmarks [21].

The results in Table II show a different trend compared to the worst-case error results in Table I. In general, the approximation pre-processor has classified a lesser percentage of faults as approximation redundant in the first category of hand crafted approximated adder designs. This has to be expected since each approximation scheme is targeted for a different error criteria, and therefore has a different sensitivity for each

TABLE II
SUMMARY OF THE APPROXIMATION-AWARE FAULT CLASSIFICATION RESULTS FOR THE BIT-FLIP ERROR

Benchmark Details		#Faults			time
Approximate adders[1]	#gates	f_{orig}	f^{bf}_{final}	f^{bf}_{Δ}(%)	sec
ACA_II_N16_Q4	225	483	400	17.18%	4s
ACA_II_N16_Q8	255	535	480	10.28%	4s
ACA_I_N16_Q4	256	530	426	19.62%	5s
ETAII_N16_Q8	255	535	480	10.28%	5s
ETAII_N16_Q4	225	483	400	17.18%	4s
GDA_St_N16_M4_P4	258	575	508	11.65%	5s
GDA_St_N16_M4_P8	280	617	197	68.07%	7s
GeAr_N16_R6_P4	263	561	200	64.35%	5s
GeAr_N16_R4_P8	261	552	199	63.95%	6s
GeAr_N16_R4_P4	255	535	480	10.28%	5s
Arithmetic designs[2]	#gates	f_{orig}	f^{bf}_{final}	f^{bf}_{Δ}(%)	sec
Han Carlson Adder*	655	1415	1202	15.05%	155s
Kogge Stone Adder*	839	1789	1699	5.03%	105s
Brent Kung Adder*	545	1178	1018	13.58%	58s
Wallace Multiplier*	641	1641	309	81.17%	52s
Array Multiplier*	610	1585	311	80.37%	55s
Dadda Multiplier*	641	1641	303	81.13%	54s
MAC unit1*	725	1821	1775	2.53%	70s
MAC unit2*	874	2104	2017	4.13%	161s
EPFL circuits[3]	#gates	f_{orig}	f^{bf}_{final}	f^{bf}_{Δ}(%)	sec
Barrel shifter*	3975	8540	3454	59.55%	61488s
Alu control unit*	178	378	178	52.91%	11s
Coding-cavlc*	885	1830	1346	26.45%	76s
Lookahead XY router*	370	739	655	11.37%	77s
Int to float converter*	296	624	293	53.04%	9s
Priority encoder*	1225	2759	1061	61.54%	87s
ISCAS circuits[4]	#gates	f_{orig}	f^{bf}_{final}	f^{bf}_{Δ}(%)	sec
c499*	577	1320	1153	12.65%	73s
c880*	527	1074	305	71.60%	31s
c1355*	575	1330	1196	10.08%	79s
c1908*	427	974	949	2.57%	30s
c2670*	931	1950	428	78.05%	396s
c3540*	1192	2657	839	68.42%	418s
c5315*	2063	4224	1648	60.98%	6224s
c6288*	2836	7048	3071	56.42%	4881s

#gates: gate count after synthesis
f_{orig}: original fault count for ATPG without bit-flip approximation
(Note: dominant and equivalent faults are excluded from this count)
f^{bf}_{final}: final fault count after approximation-aware fault classification
f^{bf}_{Δ}(%): Reduction in fault count $= \left(f_{orig} - f^{bf}_{final} \;/\; f_{orig}\right) * 100$
time: CPU time taken by approximation-aware fault classification
Benchmark sources: [1]Approximate adders from [19]
[2]Arithmetic designs [20], [3]EPFL circuits [21], [4]ISCAS circuits [22]
$*$ shows approximation technique using the public tool [15]

of these error metrics. Furthermore, these two error metrics are not correlated. As an example, a defect affecting only the most significant output bit has the same bit-flip error as that of a defect affecting the least significant output bit of the circuit. However, the worst-case errors for these respective defects are vastly different. We refer to the individual works [16], [17], [18], [19] etc., for a detailed discussion of the error criteria employed in the design of these circuits. Nevertheless, our approximation-aware fault classification tool is able to classify a significant number of faults as approximation redundant in several circuits provided in Table II.

Overall, the results confirm the applicability of our proposed methodology. Note that, in general the run-times for a SAT based ATPG flow depend mainly on the circuit complexity, size and the underlying SAT techniques. Our approach is also influenced by these factors. Therefore, improvements in SAT-based ATPG has a direct impact in our approach. It is also worth mentioning that the approximation-aware fault classification and the subsequent ATPG generation is a one time effort whereas the actual post production test of the circuit is a recurring one. Hence, the additional effort and run-times are easily justified due to high reduction in the fault count that has to be targeted for test generation.

VI. CONCLUSIONS

In this work, we presented an approximation-aware test methodology. First, we proposed a novel fault classification based on the approximation error characteristics. Further, we showed a formal methodology that can map all the faults of an approximation circuit into approximation-redundant and non-approximation faults. The approximation-redundant faults are guaranteed to have effects that are below the error threshold limits of the AC application. Hence, the subsequent ATPG generation has to target only the non-approximation faults and thereby yield can be improved significantly.

Our methodology can be easily integrated into today's standard test generation flow. Besides, the experimental results on a wide range of circuits confirm the potential and significance of our approach. Substantial reduction in fault count up to 80% is obtained depending on the concrete approximation and the error metric.

REFERENCES

[1] V. Chippa et. al, "Analysis and characterization of inherent application resilience for approximate computing," in *DAC*, 2013, pp. 1–9.
[2] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, 2016.
[3] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *ICCAD*, 2016, pp. 83:1–83:8.
[4] ——, "Precise error determination of approximated components in sequential circuits with model checking," in *DAC*, 2016, pp. 129:1–129:6.
[5] R. Venkatesan et. al, "MACACO: modeling and analysis of circuits for approximate computing," in *ICCAD*, 2011, pp. 667–673.
[6] S. Sindia and V. D. Agrawal, "Tailoring tests for functional binning of integrated circuits," in *ATS*, 2012, pp. 95–100.
[7] K. J. Lee, T. Y. Hsieh, and M. A. Breuer, "A novel test methodology based on error-rate to support error-tolerance," in *ITC*, 2005, pp. 9 pp.–1144.
[8] H. Ichihara, K. Sutoh, Y. Yoshikawa, and T. Inoue, "A practical approach to threshold test generation for error tolerant circuits," in *ATS*, 2009, pp. 171–176.
[9] K. J. Lee, T. Y. Hsieh, and M. A. Breuer, "Efficient overdetection elimination of acceptable faults for yield improvement," *TCAD*, vol. 31, pp. 754–764, 2012.
[10] I. Wali, M. Traiola, A. Virazel, P. Girard, M. Barbareschi, and A. Bosio, "Towards approximation during test of integrated circuits," in *DDECS*, 2017, pp. 28–33.
[11] ——, "Can we approximate the test of integrated circuits?" in *Workshop on Approximate Computing, WAPCO*, 2017.
[12] J. P. Roth, "Diagnosis of automata failures:a calculus and a method," vol. 10, pp. 278–281, 1966.
[13] B. Becker, R. Drechsler, S. Eggersglüß, and M. Sauer, "Recent advances in SAT-based ATPG: Non-standard fault models, multi constraints and optimization," in *DTIS*, 2014, pp. 1–10.
[14] M. Breuer, "Determining error rate in error tolerant VLSI chips," in *DELTA*, 2004, pp. 321–326.
[15] M. Soeken, D. Große, A. Chandrasekharan, and R. Drechsler, "BDD minimization for approximate computing," in *ASP-DAC*, 2016, pp. 474–479.
[16] A. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *DAC*, 2012, pp. 820–825.
[17] N. Zhu, W. L. Goh, and K. S. Yeo, "An enhanced low-power high-speed adder for error-tolerant application," in *ISIC*, 2009, pp. 69–72.
[18] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," in *ICCAD*, 2013, pp. 48–54.
[19] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *DAC*, 2015, pp. 1–6.
[20] (2016) Aoki Laboratory,Tohoku University. [Online]. Available: http://www.aoki.ecei.tohoku.ac.jp/arith
[21] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *IWLS*, 2015. [Online]. Available: http://infoscience.epfl.ch/record/207551
[22] M. C. Hansen et. al, "Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering," *D&T*, pp. 72–80, 1999.
[23] A. Biere, M. Heule, H. van Maaren, and T. Walsh, *Handbook of Satisfiability*. IOS Press, 2009.