Towards approximation during test of Integrated Circuits

Imran Wali, Marcello Traiola, Arnaud Virazel, Patrick Girard, Mario Barbareschi, Alberto Bosio

Towards Approximation during Test of Integrated Circuits

Imran Wali[3], Marcello Traiola[1], Arnaud Virazel[1], Patrick Girard[1], Mario Barbareschi[2], Alberto Bosio[1]
[1]LIRMM - University of Montpellier / CNRS - France
[2]DIETI - University of Naples Federico II - Italy
[3]University of Rennes I / INRIA/IRISA - France

***Abstract*—In the recent years, Approximate Computing (AC) has emerged as a new paradigm for energy efficient design of Integrated Circuits (ICs). AC is based on the intuitive observation that, while performing exact computation requires a high amount of resources, allowing a selective approximation or an occasional relaxation of the specification can provide significant gains in energy efficiency. This work starts from the consideration that AC-based systems can intrinsically accept the presence of faulty hardware (i.e., hardware that can produce errors). In other words, an AC-based system does not need to be built using defect-free ICs. Under this assumption, we can relax test and reliability constraints of the manufactured ICs. One of the ways to achieve this goal is to test only for a subset of faults instead of targeting all possible faults. In this way, we can reduce the manufacturing cost since we reduce the number test patterns and thus the test time. We call this approach Approximate Test (AT). The main advantage is the fact that we do not need a prior knowledge of the application. Therefore, the proposed approach can be applied to any kind of IC, reducing the test time and increasing the yield. In this work, we aim at validating the proposed AT by comparing it with a functional approach. We present preliminary results on some simple case studies. The main goal is to show that by letting some faults undetected we can save test time without having a huge impact on the application quality.**
Index terms - fault coverage; test pattern; approximate test; test generation; test complexity

I. Introduction

Today's Integrated Circuits (ICs) are starting to reach the physical limits of CMOS technology. Among the multiple challenges arising from technology nodes lower the 20 nm, we can highlight the high leakage current (i.e., high static power consumption), reduced performance gain, reduced reliability, complex manufacturing process leading to low yield and complex testing process, and extremely costly masks [1]. In other words, ICs manufactured with the latest technology nodes are less and less efficient (w.r.t. both performance and energy consumption) than forecasted by the Moore's law. Moreover, manufactured devices are becoming less and less reliable, meaning that errors can appear during the normal lifetime of a device with a higher probability than with previous technology nodes [2]. Fault tolerant mechanisms are therefore required to ensure the correct behavior of such device at the cost of extra area, power and timing overheads. Finally, process variations force the engineers to add extra guard bands (e.g., higher supply voltage or lower clock frequency than required under normal circumstances) to guarantee the correct functioning of manufactured devices.

In the last years, the Approximate Computing (AC) paradigm has been emerged [2], [3], [4]. AC is based on the intuitive observation that, while performing exact computation requires a high amount of resources, allowing selective approximation or occasional violation of the application specifications can provide gains in efficiency (i.e., less power consumption, less area, higher manufacturing yield) without significantly affecting the output quality [2], [3], [4]. This work starts from the consideration that AC-based systems can intrinsically accept the presence of faulty hardware (i.e., hardware that can produce errors) [5]. The hardware-induced errors have to be analyzed to determine their propagation through the system layers and eventually determining their impact on the final application. In other words, an AC-based system does not need to be built using defect-free ICs. Indeed, AC-based systems can manage at higher-level the errors due to defective ICs, or those errors simply do not significantly impact the final applications. Under this assumption, we can relax test and reliability constraints of the manufactured ICs. One of the ways to achieve this goal is to test only for a subset of faults instead of targeting all possible faults. In this way, we can reduce the manufacturing cost since we eventually reduce the number of test patterns and thus the test time. In the literature, some interesting works have already been published so far, targeting the test generation for a subset of faults [6], [7]. The authors propose to extract all possible faults and then classifying them into two classes: 1) benign faults - those that cause no error or an acceptable amount of errors, and 2) malignant faults - those faults that cause a significant deviation from acceptable behavior. The metric used for classifying faults into these two classes is the error magnitude [2], which is the difference between the actual value (affected by the fault) and the golden value (fault-free). This classification is exploited during the test generation.

The contribution of this work is to investigate an approach opposite w.r.t. the state-of-the-art. Instead of classifying the faults according to the final application, we propose to exploit a purely structural analysis to determine the most vulnerable circuit elements and thus generate the test patterns for these elements only. We call this approach Approximate Test (AT). The main advantage is the fact that we do not need a prior knowledge of the application. Therefore, the proposed approach can be applied to any kind of IC reducing the test time and increasing the yield. This paper presents a validation

flow for AT quality assessment. We will compare the proposed structural approach with respect to a functional one. The goal is to prove that we can be really application independent. The validation is done through experiments carried out on some simple case studies. The results aim to investigate the impact of the AT on the test length and the fault coverage. Moreover, we also show the impact on the final application by using the well-known error probability (P_ε) and the error magnitude (ε) metrics [9]. The main goal is to show that by letting some faults undetected we can save test time without having a huge impact on the application quality. The paper is organized as follows. Section II presents the flow of the proposed approach and gives details of each step. Experimental results are discussed in Section III. Finally, conclusions are given in Section IV.

II. Proposed approach

The main idea of the proposed AT approach is to let some faults untested in order to speed up the test and to increase the overall yield. The important point is that the impact of the untested faults on the final application has to be in the acceptable region (i.e., the final output quality remains acceptable to the user). The real challenge is therefore to determine what are the faults that must be tested and what are those that can be ignored during the test application process. The straightforward approach for determining the targeted faults is to act according to the functionality of the circuit. The classical example is an arithmetic circuit where it is better to guarantee that the most significant outputs are correct to narrow down the ε [2]. Unfortunately this approach cannot be adopted for all the kind of integrated circuits since it is not always easy to determine the most significant outputs. Indeed the latter are clearly strictly related to the application. In order to overcome the above issue, we propose to determine the targeted faults by using a structural analysis. In this way, we want to be independent w.r.t. the circuit function. In this work, we present a flow for validating the proposed structural analysis. The flow applies both functional and structural analyses and then it compares the quality reduction of application outputs due to the untested faults. The goal is to prove that our structural approach can lead to a quality reduction similar to the functional approach with the advantage of being application independent. Figure 1 describes the main steps of the proposed AT approach. The starting point is the circuit netlist (Original Design). The first step is the netlist analysis. Actually, two types of analysis are performed: Functional and Structural. The goal of each analysis is to rank the circuit outputs based on their significance and susceptibility respectively. The ranked set of outputs is used as input in the second step. Basically, for each output we determine the related fault list considering all the gates belonging to the corresponding fan-in cone. Every time we process an output, a fault list is created by adding the new faults (i.e., we increase the number of faults every time we process a new output). Each fault list contains the faults that must be detected.

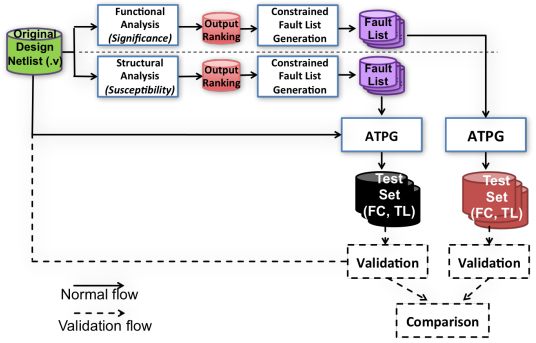

Fig. 1: Approximate Test flow

The last fault list corresponds to the whole set of faults (since we processed all the outputs), while the first one contains only the faults that belong to the first output (i.e., the most significant/susceptible). In the third step, a commercial Automatic Test Pattern Generation (ATPG) tool is used to generate a test set for every fault list. Each test set is characterized by the Fault Coverage (FC) and the Test Length (TL). The considered fault model is the Stuck-at-fault model. The test patterns are generated by using the compaction and the random-fill options. The fourth step is the validation of each generated test set. The validation is carried out through a fault injection campaign. We inject the untested faults into the circuit netlist and, we simulate the execution of the application. The application outputs are collected and the deviation between the faulty and the fault-free outputs is measured with two well-known metrics: Error Probability (P_ε) and the Error Magnitude (ε) that will be formally defined later. The two metrics quantify the output degradation. The last step compares the output degradation obtained by using the structural and the functional analysis respectively. The knowledge of the circuit functionality (i.e., the application) is mandatory if we aim to understand which outputs cannot be faulty in order to achieve an acceptable output degradation (i.e., accordingly to the quality requirements). The goal of the comparison step is to prove that, by considering only structural information, we can achieve an output degradation as acceptable as by using the functional analysis. Next subsections provide details on each step of the proposed flow.

A. Functional Analysis - Significance

The functional analysis aims at determining the targeted faults accordingly to the functionality of the circuit. As already discussed, this approach works when the application is known. For instance, when considering digital signal processing ICs, outputs can be easily ranked depending on their weight. The Most Significant Bit (MSiB) of the output data word is the one having the most significance while the Least Significant Bit (LSiB) has the lowest significance on the computed result. Thus, to reduce as much as possible the deviation between the faulty and fault-free outputs, we have to guarantee that the MSiBs are correct, therefore we have to test for all the faults affecting these outputs. The output of this step is the

ranked list of the circuit outputs (from MSiB down to LSiB). The test engineer can select how many (and which) outputs have to be "tested" in order to guarantee an acceptable output degradation (i.e., accordingly to the quality requirements).

B. Structural Analysis - Susceptibility

Conversely to the functional analysis, we propose to determine the critical faults (i.e., ones that must be targeted) by looking only at the circuit structure. In this case, the main idea is to analyze each primary output of the circuit to determine its susceptibility as described in [8]. The output susceptibility analysis is based on the fact that not all outputs of a circuit have the same susceptibility, which is a function of the number of nodes in its fan-in logic cone. It exploits the structural properties of the output fan-in cone to get their relative susceptibility estimates. In other words, we aim identifying the outputs more likely to be affected by the presence of faults. Compared to the significance analysis, the susceptibility analysis is only related to the circuit structure. No functional information is required to compute it and thus it can be applied to any digital circuits. Algorithm 1 shows the pseudo-code of the susceptibility analysis methodology. The algorithm starts by reading the pre-place-and-route netlist of the design. Then, it forms groups F_j of all fan-in cells for each circuit output O_j. Once groups are formed the weight W_j of each fan-in cone is calculated by adding together the weights of all cells in the corresponding fan-in cone group. According to the hypothesis that forms the basis of this methodology, cell weight is the number of inputs and outputs of that cell. Ranks are assigned to each output on the basis of their fan-in cone weight using a sort function shown in line 15 of Algorithm 1.

```
read(netlist);
// Group all fan-in cone cells together for
   each output node
foreach O_j do
  | F_j ← O_j.get_fanin();
end
// Get weight of fanin cone of each output
foreach O_j do
  | foreach C_i do
  | | if C_i ∈ F_j then
  | | | W_j ← W_j + C_i.get_pins();
  | | end
  | end
end
// Sort outputs on the basis of their fanin
   cone weight
sort(O_j,F_j,W_j);
```

Algorithm 1: Output susceptibility analysis

The algorithm is further explained through its application to a simple example circuit shown in Figure 2. The shaded regions mark the boundaries of the two output fan-in cones. The weight parameter (W_i) is given on the top of each gate. The fan-in cones weight (S_j) given on the right of corresponding output is found to be 14 and 10 for O_1 and O_2 respectively. According to these figures we can infer that detecting the faults affecting the fan-in cone of O_1 significantly impacts the fault coverage while detecting the remaining faults slightly improves the fault coverage. As for the significance analysis, the output of this step is the ranking of the circuit outputs from the Most Susceptible Bit (MSuB) to the Least Susceptible Bit (LSuB). As in the example of Figure 2 they are O_1 and O_2 respectively.

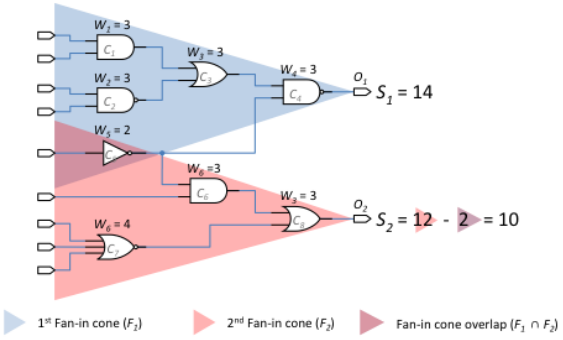

Fig. 2: Application of the susceptibility analysis

C. Constrained Fault List Generation

This step takes as input the results of the two previous analyses (Functional and Structural). Algorithm 2 shows the pseudo-code of the fault list generation. The algorithm starts from the list of ranked circuit outputs (outputs-list). Then, for each output it determines the related fan-in cone cells (line 4). From the fan-in cones cells, it extracts the associated stuck-at-faults and then, it adds the extracted faults to the Fault Universe (FU) variable (line 5). Thus, during each iteration, FU is incremented by adding the faults affecting the current output. Before moving to the next iteration, it prints the FU that is the Fault List associated to the processed O_j. From the pseudo-code we can thus define the fault list of O_j as the set of faults affecting the circuits outputs from O_0 to O_j. Once again O_0 corresponds to the most significant/susceptible output.

```
GenerateFaultList(outputs_list);
FU ← ∅ ;
// Process each output j from outputs_list
foreach O_j do
  | // Get all fan-in cone cells for the
  |    current output
  | Fan-In ← O_j.get_fanin();
  | // Add faults from Fan-in to the FU
  | FU ← FU ∪ Fan-In.get_faults();
  | // Print Fault List associated to the
  |    current output
  | FU.print();
end
```

Algorithm 2: Fault List generation

Let us come back to the example shown in Figure 2. For each output, we determine the collapsed stuck-at-fault list by applying the algorithm in Algorithm 2.

Figure 3 graphically shows the collapsed fault list obtained by considering the fan-in cone of O_1. It is composed of 9 stuck-at-faults. These 9 faults are added to the FU and they compose the fault list associated to O_1.

Figure 4 graphically shows the collapsed fault list obtained by considering the fan-in cone of O_2. It is composed of 7

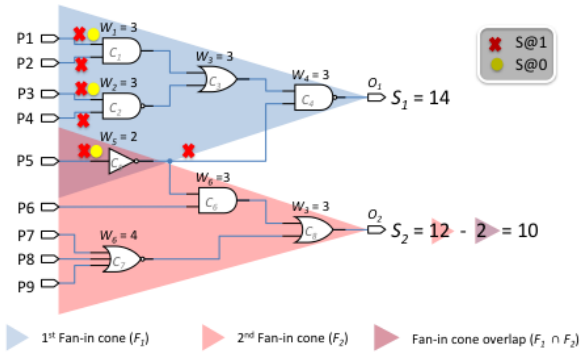

Fig. 3: Fault List generation from O_1

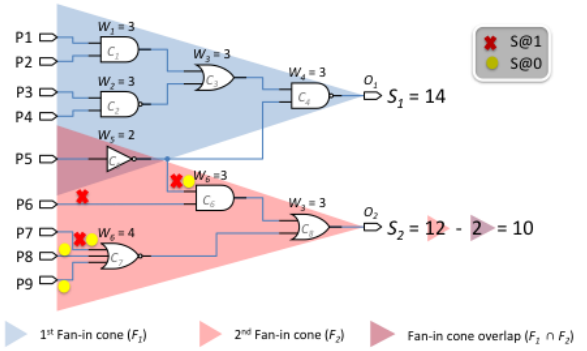

Fig. 4: Fault List generation from O_2

stuck-at-faults. These 7 faults are further added to FU that will now be composed of 16 faults. These faults compose the fault list associated to O_2. We detailed this step since it is important to note the fan-out branch at the output of the invert C5. In the fault list from O_1 the faults affecting the inverter have been considered. However, in O_2 we must add the faults affecting the branch impacting on the O_2 fan-in cone itself. Otherwise those faults will never be targeted during the ATPG.

D. ATPG

For this step of the AT flow, a commercial ATPG tool, with default options (i.e., fault dropping, dynamic compaction and random fill), is used to generate the test patterns. The Fault Coverage (FC) and the Test Length (TL) are two important outputs since they represent the effectiveness of the proposed AT compared to a deterministic one. We also keep the patterns list for the validation step. Figure 5 gives an example of the test generation. First of all, the target fault is the S@1 affecting the primary input P1 (circled red cross in the figure). The ATPG determines the logical values to be applied to the first 5 primary inputs (from P1 to P5) for sensitizing and propagating the fault effect through the circuit gates to reach O_1. The primary inputs impacting only on the O_2 fan-in cone are simply randomly filled (from P6 to P9). The logical values are shown in the Figure 5. It is clear that more than one fault is tested "for free", thanks to the input assignments. In this particular example, we test also S@0 affecting P3, S@1 affecting P6 and S@0 affecting P7. Thus, we can also detect faults affecting O_2 cone thanks to the random filling.

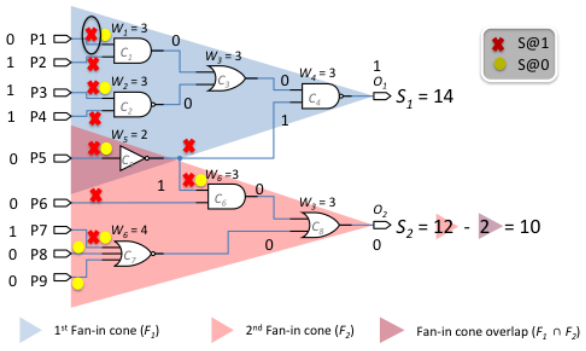

Fig. 5: Test generation example

Outputs	#. Faults	Detected Faults		FC (%)	TL
		O_1	O_2		
O_1	9	9	5	87.5	6
O_2	16	9	7	100	8

TABLE I: Example

Table I summarizes the test generation for the given example. Let us consider the first row. It reports the ATPG data executed by using the O_1 fault list. The achieved fault coverage (FC) is 87.5% and 6 test patterns are generated (TL). The column "detected faults" gives the detail per each fault list. It can be seen that targeting the O_1 fault list will result in detecting for "free" 5 faults of the O_2 fault list, thanks to the random fill. In the second row the fault list of O_2 is used. In this case, the ATPG targets all the possible 16 faults and it achieves the 100% of fault coverage. The number of test patterns is thus higher.

E. Validation

The validation is carried out through a fault injection campaign. We inject the untested faults into the circuit netlist and, we simulate the execution of the application. The application outputs are collected and the deviation between the faulty and the fault-free outputs is measured with two well-known metrics: Error Probability (P_ε) and the Error Magnitude (ε):

- P_ε represents the ratio of erroneous responses among the total responses that the circuit can produce;
- ε represents the absolute deviation of output result from the accurate one.

P_ε can be computed for any digital circuit while ε computation is only possible when the application is known. In this work, we do not consider a particular application. Thus, to be able to compute the ε, we will use arithmetic circuits for which it is easy to identify the weight of each output and consequently it is easy to compute the difference between the faulty and fault-free outputs. To be precise, we compute both metrics by running an exhaustive fault injection campaign. All possible stuck-at-faults are injected (one after the other) and the exhaustive pattern list is simulated. It is worth to mention that this exhaustive fault injection can only be applied for small circuits (i.e., with a limited number of inputs and outputs).

Responses are then compared to the golden ones to compute P_ε metric. ε metric is obtained in the same way but the difference between the simulated fault injection and the golden responses is obtained with the help of the output ranking computed during the functional analysis. Once again this is done only for arithmetic circuits for which we can determine the significance of the outputs without the knowledge of the application.

III. Experimental results

The AT flow has been validated on three case studies. All the circuits have been synthesized with a 45nm technology library [10]. Table II reports the main characteristics of each circuit in terms of number of primary inputs (#. PIs), primary outputs (#. POs) and logic gates (#. Gates). All the case studies are purely combinational circuits. The first two are arithmetic circuits able to compute 4-bit sum, subtraction and logic operation. Moreover, the ALU2 can also compute comparisons between the two inputs (i.e., is equal to, greater than and less than). The last circuit is the c432 from the ISCAS'85 benchmarks suite [11]. We selected these benchmarks in order to have two cases for which both the functional and structural analyses can be applied, while for the last one, only the structural analysis is applied.

Circuit	#. PIs	#. POs	#. Gates
ALU	11	4	51
ALU2	12	7	127
c432	36	7	160

TABLE II: Case Studies

The goal of the validation is to show that, when the workload is not known, the structural analysis can be used instead of the functional one. Moreover, we want to show that the error obtained by the structural analysis is comparable with the one obtained through the functional one. Thus, the proposed structural approach can be adapted to any kind of circuits without knowledge of the application. In the next sub-sections we present the results for each case study.

A. ALU

The ALU circuit can compute the standard arithmetic and boolean operation over the two 4-bit inputs. Table III reports the results obtained by applying the AT flow described in the Section II. Since this case study is an arithmetic circuit, we applied both the functional and structural analyses. In this particular case, the order provided by the two analyses is the same: z_3, z_2, z_1 and z_0. This is because the MSiB (z_3) is the one having the most important fan-in cone and thus it also corresponds to the MSuB. The first column of Table III gives the number of primary outputs considered in the second step for generating the fault list and, consequently the test patterns. Let us remember that we selected the primary output for the fault list and the test pattern generation accordingly with their ranking (i.e., from the MSiB/MSuB down to the LSiB/LSuB). Looking at first row, we can see that if we consider only one output (i.e., the fault list contains only the faults affecting the fan-in cone of that primary output), we can achieve a high stuck-at-fault coverage (96.58%) with a probability of 0.199 to have a failure at the output (i.e., due to the untested faults). For this case, the average error magnitude (ε_{avg}) is 0.580 (i.e., the average deviation from the expected output in the presence of faults) and maximum error magnitude (ε_{max}) is 7 (i.e., the highest difference observed between the fault-free circuit and the faulty one). It is worth to consider that a maximum deviation of 7 in the result correspond to a maximum error of 46.67% (i.e., 7 over 15).

Outputs	FC (%)	TL	P_ε	ε_{avg}	ε_{max}
1	96.58	19	0.199	0.580	7
2	98.16	21	0.140	0.243	3
3	98.95	23	0.060	0.060	1
4	99.47	24	0	0	0

TABLE III: All results

Then, by adding the remaining outputs we slightly increase the fault coverage, to reach at the end 99.47%. Considering the test length, we can notice that the number of test patterns varies from 19 up to 24. Thus, only considering one output we can reduce the test length of about 21%. Depending on the final system quality, the test engineer can therefore select the appropriate set of primary output leading to the desired trade-off between fault coverage, test length, P_ε and ε_{max}. Note that the last line in Table III represents the case of a standard deterministic test where all stuck-at-faults are considered during the test generation (i.e., the upper bound of the FC and TL).

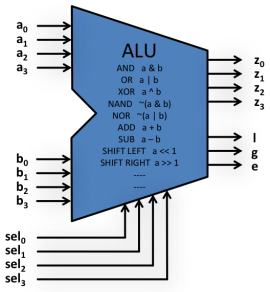

Fig. 6: ALU2 block diagram

B. ALU2

Figure 6 shows the block diagram of the ALU2. This circuit can compute the standard arithmetic and boolean operation over the two 4-bit inputs and, in addition, comparison between the inputs. Even if this circuit is quite simple in terms of netlist and number of inputs and outputs (as reported in Table II), it is interesting since the functional analysis is not so obvious. Clearly, z_3 is still the MSiB of the data outputs, but what is the weight of the logical outputs (l, g, and e)? Once again, this question can only be answered by the knowledge of the application. In the following we consider two possible scenarios:

- S1: the logical outputs are more significant than the data outputs. Thus the Functional analysis will provide the rank l, g, e, z_3, z_2, z_1 and z_0.

- S2: the logical outputs are less significant than the data outputs. Thus the Functional analysis will provide the rank z_3, z_2, z_1, z_0, l, g and e.

Let us consider S1 as the target scenario for our experiment, so that we can observe what happen when the output ranking is different for functional and structural analysis.

Outputs	FC(%)	TL	P_ε	ε_{avg}	ε_{max}
1	76.19	27	0.287	1.551	15
2	76.62	29	0.257	1.328	15
3	77.16	31	0.250	1.271	15
4	94.83	55	0.192	0.536	7
5	97.52	62	0.145	0.245	3
6	98.28	66	0.082	0.082	1
7	100.00	75	0	0	0

TABLE IV: Functional Analysis: Scenario S1

Outputs	FC(%)	TL	P_ε	ε_{avg}	ε_{max}
1	85.56	38	0.256	0.592	7
2	88.36	45	0.209	0.302	3
3	89.76	50	0.147	0.141	1
4	93.10	57	0.067	0.063	1
5	99.35	70	0.039	0.038	1
6	99.89	71	0.007	0.007	1
7	100.00	72	0	0	0

TABLE V: Structural Analysis: Scenario S2

Tables IV and V report the results obtained by using only the functional and the structural analysis respectively. We can immediately compare the tables. Let us consider the first row of both the cases. As for the functional analysis, when considering only one output and its related fan-in cone set of stuck-at-faults, we achieve 76% of FC. Conversely, when the output is provided by the structural analysis we can reach 85% of FC. There is a difference of 9% of FC. The P_ε and ε_{avg}, ε_{max} metrics also show a significant difference between the two cases. The ones associated to the structural analysis are lower than those obtained by using the functional analysis. Especially the ε_{avg} and ε_{max} are reduced from 1.55 to 0.59 and from 15 down to 7 respectively, which correspond to a 2.6 times reduction in ε_{avg} and a 2.1 times reduction in ε_{max} when comparing our structural analysis with the functional one. Finally, both analyses lead us to a maximum reduction of about 45% of test length compared to the upper bound case (i.e. the last line of the tables where all stuck-at-faults are considered during the test generation). Hence, the test engineer can select the appropriate analysis leading to the desired trade-off between fault coverage, test length, P_ε and ε metrics value. Moreover, in this case, the structural analysis turns out to be even better than the functional one. Due to lack of results, we cannot claim that it is always true, but it is a non-intuitive result: only considering the structural information we can improve the coverage of the functionalities of the target circuit since both P_ε and ε are reduced.

C. c432

The last case study is the public available benchmark c432 [11]. As reported in Table II it has 36 primary inputs. This case study has been selected in order to put the study in a context close to real cases where the P_ε and ε can only be computed with a given application. Thus, the Table VI does not provide any value for P_ε and ε. Reported results are indeed interesting: by considering only one output (i.e., the MSuB), we can achieve the 97.32% of FC. On the other hand, if the complete fault list is generated the FC is 97.88%. So we simply loose 0.5% of fault coverage. Looking at the test complexity, we can reduce it by 4% if we consider only the MSuB output for the fault list generation.

Outputs	FC(%)	TL
1	97.32%	65
2	97.77%	67
3 - 7	97.88%	68

TABLE VI: C432 Results

IV. Conclusion

The contribution of this work is to investigate an approach opposite w.r.t. the state of the art. Instead of classifying the faults according to a given application, we exploit a structural analysis to determine the most vulnerable circuit elements and thus generate test patterns for these elements. Preliminary results indicate that the resulting method, called Approximate Test can really lead to provide benefits in terms of test time reduction. Test engineer can select the desired trade-off between quality and test complexity. The proposed structural analysis methodology seems interesting and promising step toward making the AT applicable to any kind of circuits. Future works are mainly devoted to prove the effectiveness of the approach performing experiments on a wider set of circuits and to extend the structural approach to other fault models.

References

[1] G. Gielen, et al., "Emerging yield and reliability challenges in nanometer CMOS technologies", in Proceedings of the Conference on Design, Automation and Test in Europe, 2008, pp. 1322-1327

[2] S. Mittal, "A Survey of Techniques for Approximate Computing", ACM Computing Surveys (CSUR), Vol. 48, no. 4, 2016, pp. 1-33.

[3] Q. Xu, T. Mytkowicz and N. S. Kim, "Approximate Computing: A Survey," in IEEE Design & Test, vol. 33, no. 1, 2016, pp. 8-22.

[4] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," 18th IEEE European Test Symposium (ETS), Avignon, 2013, pp. 1-6.

[5] V. Chippa et al., "Analysis and characterization of inherent application resilience for approximate computing", in Proc. 50th ACM/EDAC/IEEE Design Automation Conference (DAC'13), 2013, pp. 1-9.

[6] K.-J. Lee, T.-Y. Hsieh, and M. A. Breuer, "Efficient Overdetection Elimination of Acceptable Faults for Yield Improvement," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 31, no. 5, 2012, pp. 754-764.

[7] S. Sindia, V. D. Agrawal, "Tailoring Tests for Functional Binning of Integrated Circuits", IEEE 21st Asian Test Symposium, 2012, pp. 95-100.

[8] I. Wali, B. Deveautour, A. Virazel, A. Bosio, P. Girard, M. Sonza Reorda, "A Low-cost Susceptibility Analysis Methodology to Selectively Harden Logic Circuits", IEEE 21st European Test Symposium, 2016, pp. 1-2.

[9] A. Momeni, J. Han, P. Montuschi and F. Lombardi, "Design and Analysis of Approximate Compressors for Multiplication," in IEEE Transactions on Computers, vol. 64, no. 4, 2015, pp. 984-994,.

[10] Nangate Inc., "45nm open cell library", http://www.nangate.com. Last visit: feb. 2017.

[11] https://filebox.ece.vt.edu/~mhsiao/iscas85.html. Last visit: feb. 2017.