Sources of Bias in EDA Tools and Its Influence

Petr Fišer, Jan Schmidt, Jiří Balcárek
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
fiserp@fit.cvut.cz, schmidt@fit.cvut.cz; balcaji2@fit.cvut.cz

Abstract— **In this paper we present an experimental analysis of robustness of Electronic Design Automation (EDA) tools, with respect to different seemingly unimportant aspects (bias) introduced by the designer, "from outside". The algorithms employed in EDA tools should be immune to these completely, since such aspects do not carry any useful information – source files differing in these aspects are semantically equivalent. However, we show that most of the studied tools are seriously sensitive here, much more than ever reported. The results indicate, that experiments conducted to evaluate the performance of EDA tools must take such behavior into consideration. Also the notion of a benchmark is questioned.**

Keywords- logic synthesis, EDA tools, randomness, robustness

I. Introduction

Because of high complexities of present designs, using exact (optimum) logic synthesis and optimization algorithms in Electronic Design Automation (EDA) tools is not feasible. Therefore, approximate heuristic algorithms must be used in practice [1]. These heuristics can be influenced by many external aspects and results of different quality can be produced under different circumstances. In this paper we will show an experimental evaluation of several tools, to illustrate that the situation is much more severe than previously reported [2], [3].

To cover maximum of the EDA design flow, we will examine several logic design and optimization tools, both academic [4], [5], [6], [7] and commercial ones, one Automated Test Patterns Generation (ATPG) algorithm [8], and two test compression tools [9], [10].

To give an example of the studied phenomenon, it has been shown that many synthesis processes are *not immune* to the structure of the source file, like the ordering of variables in the file header [11] or small modifications in RTL statements [2]. Results of different quality were obtained by only modifying the source file header or renaming variables.

Actually, the source data for the tools (be it source circuit descriptions, test vectors, etc.) carry additional information, like the ordering of coordinate statements, that should not be respected by the algorithms, in sense of invariance of results. Source files carrying different pieces of information of such a kind are *semantically equivalent*, e.g., they represent equally described circuits, or equal sets of test vectors. Clearly enough, the result quality should not be influenced by such information. Unfortunately, this is far from true in practice.

From the practical point of view, the designer (RTL code programmer) should not care about this information. Therefore, the designer generates such pieces of code *unconsciously randomly*, because of he/she considers this information completely unimportant. We will refer to this phenomenon as *unpredictable external bias*. As a result, deterministic algorithms used in logic design (which they typically are) may become stochastic, with respect to that external bias.

II. Sources of External Bias

In most of (inexact) algorithms employed in EDA tools some heuristic function is typically used to guide the search. Even though the heuristics are deterministic, there can appear *multiple equally valued choices*. In such situations, the first occurrence is typically taken. Note that these choices are equally valued just at the point of decision, however, they will most likely influence the subsequent decisions. Therefore, different results could be produced, if different decisions were taken *in any step*, without affecting the principles of the algorithms. In other words, the results obtained by one single deterministic algorithm heavily depend on its software implementation.

Authors of [3] also noticed the fact, that algorithms used in synthesis tools (particularly ABC [5], [6]) can have such multiple decisions and aptly call this phenomenon "*CAD algorithm noise*". They have modified some basic ABC algorithms to perform equally valued decisions *randomly*, to achieve possibly different solutions. As expected, these solutions were also differing in quality, in units of percent.

Although these two notions – randomness introduced externally and internally – seem to be different problems, they have equal consequences.

Several means of external bias (perturbations) were described in [2]. Essentially, two kinds of external bias can be identified: *ordering* of coordinate statements and *names* of identifiers (variables). The coordinate statements could be the port definitions, instantiations of gates, test vectors, fault list, etc. Depending on these, heuristics perform differently, producing different results. Also the variables names impact some algorithms.

III. Experimental Observations

The first experimental evaluation of the influence of perturbations in the source file has been presented in [2]. The authors have shown that modifying the source Verilog file, so that its functional equivalence is preserved, but some constructs are changed, yields synthesis results differing in units of percent (2.5% in area and 4% in slack on average, 13% at most). This is approximately what the contemporary progress in improving the synthesis tools is. Therefore, they have rendered the experimental evaluation of tools inadequate, when performed in the form it has been performed until now. They also offered a tool introducing these perturbations into the source files, to help to perform relevant experimental evaluations.

For the experimental evaluation in [2], only two commercial tools and only a small number of benchmark circuits (33) were used. We have performed a much more exhaustive evaluation, and we will show in this section, that the reality is much worse than as reported in [2] and [3].

A. ESPRESSO

First, we have investigated an impact of external bias to the state-of-the-art SOP (sum-of-products) minimizer Espresso [4].

We have processed 148 MCNC benchmark circuits [12] by Espresso and Espresso-Exact, 10,000-times each, while the input/output variables were randomly permuted in the source PLA matrix [4]. The numbers of literals in the minimized solutions were measured. The average and maximum (over the 148 circuits) size differences between minimum and maximum values are reported in TABLE I. Even though the average size differences are acceptable, rather striking maximum size differences (up to 43%) can be observed. Even the numbers of literals obtained by Espresso-Exact differ, since Espresso-Exact guarantees minimality of the number of terms only, nothing is guaranteed for literals.

TABLE I. ESPRESSO RESULTS

	Espresso	Espresso-Exact
Permuted inputs	1.51% (max. 34.90%)	0.02% (max. 0.63%)
Permuted outs.	1.04% (max. 11.82%)	0.23% (max. 6.06%)
Permuted both	2.11% (max. 42.95%)	0.24% (max. 6.06%)

B. ABC and SIS

ABC [5], [6] is the current state-of-the-art academic logic synthesis and optimization tool, the follower of SIS [7]. It comprises both combinational and sequential synthesis, mapping to standard cells or FPGA look-up tables (LUTs), verification, and many other features. It is controlled by *commands*. A sequence of commands constitute a synthesis *script*. Each command implements a particular algorithm.

For the purpose of this paper, we will present results of only complete synthesis scripts, for standard cells mapping ("*strash; dch; map*") and 4-LUT mapping ("*strash; dch; if; mfs*"), since they comprise of all the algorithms implemented for the suggested state-of-the-art synthesis.

The same experiment was performed for two SIS [7] synthesis scripts, "*script.rugged*" and "*script.algebraic*" followed by mapping into standard cells, the MCNC library [7].

The experiments were conducted as follows: 228 circuits from the IWLS and LGSynth benchmarks sets [12], [13] were processed. Given a benchmark, its inputs and/or outputs were randomly permuted in the source file, and the resulting design size (gates, LUTs) was measured. This was conducted 1,000 times for each circuit. The average and maximum size differences are reported in TABLE II. and TABLE III.

We can see that almost all the average size differences are higher than 10% and that maximum values reach impressive values. This is much worse than reported in [2]; the result quality fluctuations caused by external bias surpass anything expected.

TABLE II. ABC RESULTS

	strash; dch; map	strash; dch; if; mfs
Permuted inputs	8.67% (max. 74.38%)	11.50% (max. 92.14%)
Permuted outs.	10.52% (max. 70.47%)	12.60% (max. 85.42%)
Permuted both	13.40% (max. 86.27%)	14.81% (max. 95.07%)

TABLE III. SIS RESULTS

	script_rugged; map	script_algebraic; map
Permuted inputs	7.50% (max. 50.19%)	5.25% (max. 30.77%)
Permuted outs.	3.48% (max. 22.03%)	1.83% (max. 14.29%)
Permuted both	8.34% (max. 50.19%)	6.21% (max. 30.77%)

In order to emphasize how serious the lack of robustness of ABC is, we will thoroughly analyze the most striking example, the *cordic* circuit [13], the one responsible for the 95.07% maximum size difference in the LUT mapping script. Particularly, the solutions spun from 27 to 687 LUTs.

For higher precision, we have synthesized the circuit using 100,000 different random orderings of variables (both input and output) and measured the frequencies of occurrence of solutions of different quality (number of LUTs). The histogram is shown in Figure 1.

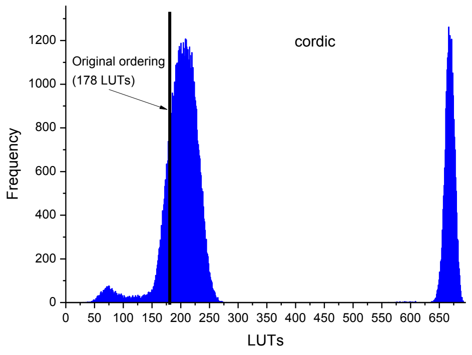

Figure 1. Distribution of solutions – *cordic*

We can observe two completely isolated regions. There are apparently two or more classes of similar implementations (similar in size, probably similar in structure too), which synthesis produces depending on the ordering of variables.

C. Commercial FPGA Synthesis Tools

Dependency of the result quality on the ordering of variables was observed in commercial FPGA design tools too. Two tools were studied and both were found to be very sensitive to the structure of the RTL statements. Surprisingly enough, the tools were also sensitive to a mere reordering of the gates instantiation, i.e., coordinate statements in the VHDL code, which was not the case of any examined process in ABC and SIS. One of the tools was also really seriously sensitive to altering signal names.

The experiment started with netlist descriptions and after permuting the variables and nodes (and randomly renaming signals) in the source file, each benchmark was converted to VHDL and submitted to synthesis and mapping. The numbers of 4-LUTs in the results were measured. Summary results of the 228 benchmarks [12], [13] are shown in TABLE IV. Again, average and maximum differences in the obtained LUT counts are shown, 1,000 random permutations were measured.

We can see that the average differences are up to 9%, whereas the maximum reaches 66.6%.

TABLE IV. Commercial FPGA Synthesis Tools

	Tool #1	Tool #2
Permuted inputs	0.0% (max. 0.0%)	4.7% (max. 43.6%)
Permuted outs.	0.0% (max. 0.0%)	5.6% (max. 52.2%)
Permuted nodes	0.2% (max. 15.8%)	3.4% (max. 38.8%)
Permuted all	0.2% (max. 17.3%)	9.2% (max. 66.6%)
Renamed signals	9.6% (max. 50.92%)	0.0% (max. 0.0%)

D. *Atalanta ATPG Tool*

As for test generation, we have exercised an academic ATPG tool Atalanta from Virginia Tech [8]. The source file to Atalanta is a circuit netlist description. In this file, we have reordered the input and output ports definitions, permuted nodes (gate definitions) and also renamed the signals. The resulting test (both compacted and non-compacted) lengths were measured.

The results are shown in TABLE V. The experiment was performed on a subset of 37 ISCAS benchmarks [14], [15], 1,000 random permutations for each.

A significant dependency of the resulting test size on the ordering of all the statements has been observed. Only the non-compacted test generation was surprisingly not sensitive to ordering of input variables. No sensitivity to signal names was observed, therefore this record is not present in the table.

TABLE V. Atalanta ATPG

	Compacted	Non-compacted
Permuted inputs	17.6% (max. 35.9%)	0.0% (max. 0.0%)
Permuted outs.	2.1% (max. 11.4%)	3.7% (max. 35.9%)
Permuted nodes	2.9% (max. 16.24%)	4.1% (max. 17.7%)
Permuted all	18.0% (max. 34.6%)	6.0% (max. 32.8%)

E. *Test Compression: RESPIN*

The RESPIN algorithm [9] will be the first representative of test compression algorithms in this study. It accepts a set of test patterns as input and produces the compressed test sequence. The compression is achieved by patterns overlapping. For details see [9].

There are two aspects influencing the algorithm run: the order of test vectors submitted as input and the initial test pattern (see [9]). In our implementation the initial pattern is set as the first test pattern. Therefore, reordering of test patterns also simulates the influence of the initial pattern. Hence, we will study only the influence of permutation of test patterns. Non-compacted test patterns obtained by Atalanta [8] were used, 52 ISCAS [14], [15] and ITC'99 [16] benchmarks were processed, with 10,000 random permutations of test vectors.

As a result, a significant sensitivity to test patterns ordering was observed. Particularly, the average difference of the compressed bitstream length was 28%, with the maximum of 54%.

F. *SAT-Compress Test Compression Tool*

The second tested test compression tool was SAT-Compress [10]. The input to SAT-Compress is the circuit description (netlist) and it directly generates the compressed test sequence. SAT-Compress does not operate with pre-computed test patterns; the test is stored implicitly, as a set of SAT instances. The compression principles are similar to RESPIN, only it sequentially processes *faults* instead of *test vectors*.

Basically, there are three aspects that may theoretically bias the algorithm progress: 1) the ordering of the fault-list, 2) the initial test pattern, 3) the structure of the netlist, even together with the variables ordering. The latter aspect comes from the fact, that SAT instances are generated from the netlist, and these are then submitted to a SAT solver [17]. Different netlist structures directly induce different SAT instances, which may influence the SAT-solver execution and different SAT solutions (used as test vectors) may be obtained.

The results of the experiment are shown in TABLE VI. ISCAS [14], [15] and ITC'99 [16] benchmarks were processed.

TABLE VI. SAT-Compress test compression

	Test sequence
Permuted inputs	59.8% (max. 73.7%)
Permuted outputs	33.8% (max. 64.5%)
Permuted nodes	23.7% (max. 45.8%)
Permuted faults	37.1% (max. 64.2%)
Random init. vect.	38.0% (max. 65.7%)
Permuted all	60.3% (max. 76.4%)

IV. Significance

EDA tools are heuristics with enormous search space. Although some of the numbers in previous tables are alarming, sub-optimal performance could be considered natural for them.

Yet benchmarking is the principal process in their development, to the extent that they can be considered social constructs. Let us return to the idea expressed in [2] and see how the observed irregularities influence benchmark comparison.

The tables below give the number of benchmarks where the tool in question can deviate by given amount with given probability, with respect to the initial ordering of variables. For example, ABC with the "*strash; dch; if; mfs*" (LUT-mapping) script gives results worse by at least 5% with 20% probability on 40 benchmarks, when input and output variables are randomly permuted (when compared to the initial ordering).

In the TABLE VII. and TABLE VIII. experiments, both input and output variables were randomly permuted, in the TABLE IX. case test vectors were reordered.

The results show variance exceeding the accuracy required to compare EDA algorithms. Most alarmingly, there seems to be no universal countermeasure. The sources of bias differ from tool to tool; in some cases (commercial Tool #1), new and unexpected sources appear. Even with all sources detected, a humble comparison requires an effort 3-4 orders of magnitude greater than practiced today.

TABLE VII. BENCHMARKS WITH GIVEN DEVIATION, ABC LUT

Probability	< -20%	< -5%	> +5%	> +20%
5%	7	72	73	8
10%	4	56	58	7
20%	2	38	40	4
50%	1	15	17	1

TABLE VIII. BENCHMARKS WITH GIVEN DEVIATION, TOOL #1

Probability	< -20%	< -5%	> +5%	> +20%
5%	131	146	43	4
10%	9	45	34	4
20%	4	30	25	3
50%	3	15	11	0

TABLE IX. BENCHMARKS WITH GIVEN DEVIATION, RESPIN

Probability	< -20%	< -5%	> +5%	> +20%
5%	2	30	30	2
10%	1	28	24	1
20%	1	18	16	1
50%	0	8	8	1

Although we can measure circuits by a number of metrics, we still do not know what "practical circuit" means. We have to rely on "genuine" benchmarks. It can be argued that the tools have adapted to the declaration ordering, signal names etc. used by real designers.

Further, more thorough experiments have shown that the initial ordering is *perfectly random* when the whole benchmark set is considered, for all the tested tools – average result size differences over random permutations and all the measured benchmarks in a given benchmark set differed from results obtained using the initial ordering no more than by units per thousand.

TABLE X. shows this in more detail. Although there are circuits where better but not worse results can be obtained (and vice versa), the benchmark set as a whole seems to be balanced in this respect.

TABLE X. OPPORTUNITIES FOR BETTER AND WORSE RESULTS

< -5% @ 10% prob.	> +5% @ 10% prob.	Tool#1	Tool#2	ABC LUT
no	no	115	187	128
no	yes	25	3	42
yes	no	36	0	40
yes	yes	9	0	16

V. CONCLUSIONS

We have presented an experimental evaluation of robustness of several EDA tools, both academic and commercial. As robustness we understand the sensitivity to outer aspects that theoretically should not influence the result, like the source file statements ordering and variable names. We call it *external bias*.

We have shown that the lack of robustness is much more severe than previously reported, and thus we emphasize the fact that the quality of EDA tools should be judged with respect to the external bias, otherwise any reported qualitative improvement/deterioration can be caused just by random bias.

Since experimental evaluations of EDA algorithms and tools rely mostly on benchmarks, we may also question the benchmark notion: is a circuit modified in the above-mentioned way still the same benchmark circuit? If so, experimental evaluation must be performed with respect to random bias.

ACKNOWLEDGMENT

This research has been supported by the grant of the Czech Technical University in Prague, SGS14/105/OHK3/1T/18.

REFERENCES

[1] S. Hassoun and T. Sasao, *Logic Synthesis and Verification*, Boston, MA, Kluwer Academic Publishers, 2002, 454 p.

[2] A. Puggelli, T. Welp, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Are Logic Synthesis Tools Robust?," in Proc. of the 48th ACM/EDAC/IEEE Design Automation Conference (DAC), 5-9 June 2011, pp. 633-638.

[3] W. Shum and J. H. Anderson, "Analyzing and predicting the impact of CAD algorithm noise on FPGA speed performance and power," in Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA'12), pp. 107-110.

[4] R.K. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis*, Boston, MA, Kluwer Academic Publishers, 1984, 192 p.

[5] Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification", http://www.eecs.berkeley.edu/~alanmi/abc/

[6] R. Brayton and A. Mishchenko, "ABC: An Academic Industrial Strength Verification Tool," in Proc. of the 22nd Intl. Conference on Computer Aided Verification, Edinburgh, July 15-19, 2010, pp. 24-40.

[7] E.M. Sentovich et al., "SIS: A System for Sequential Circuit Synthesis," Electronics Research Laboratory Memorandum No. UCB/ERL M92/41, University of California, Berkeley, CA 94720, 1992, p. 52.

[8] H.K. Lee and D.S. Ha, "Atalanta: an Efficient ATPG for Combinational Circuits," Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1999.

[9] R. Dorsch and H.-J. Wunderlich, "Reusing Scan Chains for Test Pattern Decompression," in Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 18, Issue 2, April 2002, pp. 231-240.

[10] J. Balcárek, P. Fišer, and J. Schmidt, "Techniques for SAT-based Constrained Test Pattern Generation," in Microprocessors and Microsystems, Vol. 37, Issue 2, March 2013, Elsevier, pp. 185-195.

[11] P. Fišer and J. Schmidt, "How Much Randomness Makes a Tool Randomized?," in Proc. of the 20th International Workshop on Logic and Synthesis (IWLS), San Diego, California, USA, 3. 5.6.2011, pp. 136-143.

[12] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide," Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991, p. 45.

[13] K. McElvain, "LGSynth93 Benchmark Set: Version 4.0", Mentor Graphics, May 1993.

[14] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," in Proc. of the International Symposium on Circuits and Systems, 1985, pp. 663-698.

[15] F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in Proc. of the International Symposium of Circuits and Systems, 1989, pp. 1929-1934.

[16] F. Corno, M.S. Reorda, G. Squillero, "RT-level ITC`99 benchmarks and first ATPG results," in: Proc. of the IEEE Design and Test of Computers (2000), pp. 44-53.

[17] N. Éen, N. Sorensson, "An extensible SAT-solver," in Lecture Notes in Computer, Science 2919 - Theory and Applications of Satisability Testing. Springer Verlag, Berlin Heidelberg New York (2004), pp. 333-336.