Introduction to Lethal Circuit Transformations

Petr Fišer and Jan Schmidt

Faculty of Information Technology, Dept. of Digital Design
Czech Technical University in Prague
Prague, Czech Republic

fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

Abstract. Logic optimization is a process that takes a logic circuit description (Boolean network) as an input and tries to refine it, to reduce its size and/or depth. An ideal optimization process should be able to devise an optimum implementation of a network in a reasonable time, given *any* circuit structure at the input. However, there are cases where it completely fails to produce even near-optimum solutions. Such cases are typically induced by non-standard circuit structure modifications. Surprisingly enough, such deviated structures are frequently present in standard benchmark sets too. We may only wonder whether it is an intention of the benchmarks creators, or just an unlucky coincidence. Even though synthesis tools should be primarily well suited for *practical* circuits, there is no guarantee that, e.g., a higher-level synthesis process will not generate such unlucky structures. Here we present examples of circuit transformations that lead to failure of most of state-of-the-art logic synthesis and optimization processes, both academic and commercial, and suggest actions to mitigate the disturbing effects.

INTRODUCTION

Logic synthesis and optimization [1] is a process transforming an initial circuit description (be it RTL or gate-level) into a structure suitable for the final implementation in the target technology (ASIC, FPGA).

Efficient algebraic and Boolean logic synthesis and optimization methods have been developed already in 1980s [1] and implemented in an academic synthesis tools SIS [2]. The academic state-of-the-art is now the And-Inverter Graph [3] based tool ABC [4], whose development still continues.

The synthesis process is believed to be mature and it is supposed to be efficient enough for practical applications. However, recently there emerged hints that it crucially fails for some circuits, mostly due to improper circuit descriptions given as an input [5, 6, 7]. In particular, insufficiency of logic optimization and technology mapping has been reported in [5], where artificial benchmark circuits were constructed by obscuring their original structure and enlargement by decomposition. Similar cases were reported in [6] and [7], showing inability of tools to discover obscured XOR gates. In both cases, results up to two orders of magnitude larger the optimum were produced.

Even though such unlucky circuits were artificially constructed, there are several reasons why the logic synthesis community should be concerned: 1) an ideal synthesis process should be able to efficiently cope with *any* circuit description, regardless any disturbing structural changes. 2) Unsuitable circuit descriptions may emerge in a good intention of a designer [8]. 3) Such cases appear in standard benchmark sets too [9]. We may wonder why this happens; was it intention to test capabilities of synthesis tools, or were these descriptions generated just incidentally?

We present some simple circuit transformations retaining the functional equivalence, that are "lethal" for logic synthesis, show their severity, and propose possible ways of coping with them.

CIRCUIT TRANSFORMATIONS

Several example circuit transformations will be studied in this section, and their impact on synthesis tools will be evaluated next. Unfortunately, more comprehensive results and analyses are not present due to the page limitation.

The first presented transformation is the *Network Collapsing*. Since most of the tested benchmark circuits are of multi-level origin, collapsing them into a two-level sum-of-products (SOP) form will completely destroy their original structure and hinder all the effort of their designers (in case they were originally well designed).

It is well known that the circuit size can grow exponentially with the number of its inputs by this process. We may ask whether the synthesis will be able to re-discover the original structure, or the result size will immensely grow too. Another case could be a bad description of the original circuits, where collapsing could even help.

The second transformation which can increase the circuit size exponentially is a *Transformation into BDD*. By constructing a canonical global BDD (Binary Decision Diagram [10]), the original circuit structure will be completely destroyed too. A logic network consisted of multiplexers can be directly constructed from a BDD.

Also a *Blind Decomposition into Simple Gates* will increase the circuit size, assumed that the original circuit description contained more complex gates, like XORs and multiplexers. A blind (spontaneous, uninformed) decomposition will obscure the presence of such gates, thus, again, we may ask whether the synthesis will be able to re-discover the original implementation. Note that such a transformation does not destroy the multilevel circuit structure at all; it just introduces another, misleading structural information.

Finally, the transformations can be *combined*. In particular, the network collapsing can be followed by decomposition into simple gates. Actually, exactly the same process has been used in [5] to destroy original structures of some circuits and increase their size by decomposition and introduce a misleading multi-level structure.

EXPERIMENTAL RESULTS

For the experiments we have used a mix of standard logic synthesis benchmarks coming both from academia and industry [9, 11, 12, 13, 14, 15] comprising 490 circuits of sizes ranging up to tenths of thousand FPGA look-up tables (LUTs) after synthesis. In our experiments, the source circuits have undergone the respective transformation, were submitted to the synthesis, and then the results were compared to results obtained by the synthesis of the original benchmark circuits.

The ABC '*collapse*' command was used for collapsing. For BDD transformation, the BDD manipulation package CUDD [16] was used to construct a global BDD and dump it into a network of multiplexers. SIS '*tech_decomp –a 2*' command was used to perform a blind decomposition into 2-input AND gates and inverters.

A comparison of the sizes of the original and transformed circuits from the benchmark mix, in terms of the number of literals, is shown in Figure 1 for the BDD transformation and in Figure 2 for "*collapse*" followed by "*tech_decomp*". Each dot in the graphs represents a single circuit. We can see that in most cases the size increase can be observed (72% for BDD, 79% for collapsing), sometimes by more than three and two orders of magnitude respectively (6,000- and 700-times respectively), for the two processes. However, in some cases such a transformation actually decreased the size (up to more than 80- and 32-times respectively). These were probably the cases referred to in the following section.

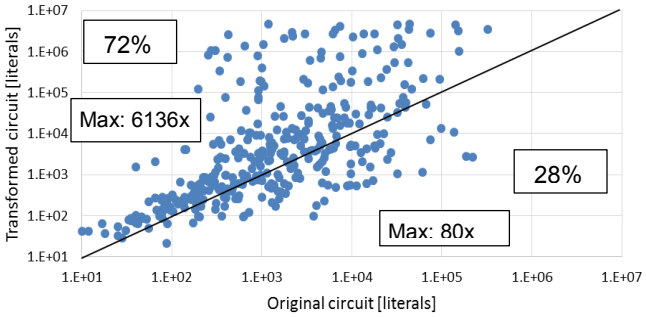

FIGURE 1. Circuit size change after transformation into BDD

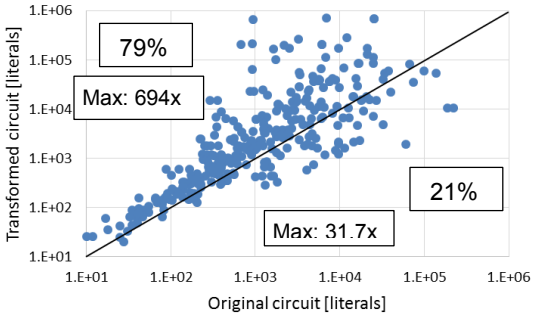

FIGURE 2. Circuit size change after "*collapse*" and "*tech_decomp*"

The impact of the studied circuit transformations to both academic and industrial tools will be evaluated next. The state-of-the-art representative of academic logic synthesis tools is ABC [4]. Results of only LUT mapping will be shown in this paper, however, similar results were obtained from standard cells mapping as well. In particular, the best known LUT synthesis script '*strash; dch; if; mfs*' suggested by ABC authors [4] was used, run iteratively

(repetitively) 20 iterations (chosen as a good compromise between run-time and result quality) and a "synthesis overkill", 5000 iterations of the same script with permutations applied in each iteration [17], to show results that can ever be obtained by conventional synthesis, at expense of extreme run-time (up to weeks). Next, two commercial LUT-mapping tools (#1, #2) were exercised.

The synthesis results are shown in Figure 3 and Figure 4, for all the four processes and two transformations. Ideally, all the solutions should be situated at the diagonal line – the case where equal solutions are produced by synthesis, regardless any transformation. We can see that this is far from the truth; all the tools have been found sensitive to the transformation, both in positive and negative ways. Generally, the tools are sensitive to the size of the input description in this case; the "bigger" circuit description is submitted to synthesis, the bigger will the result be.

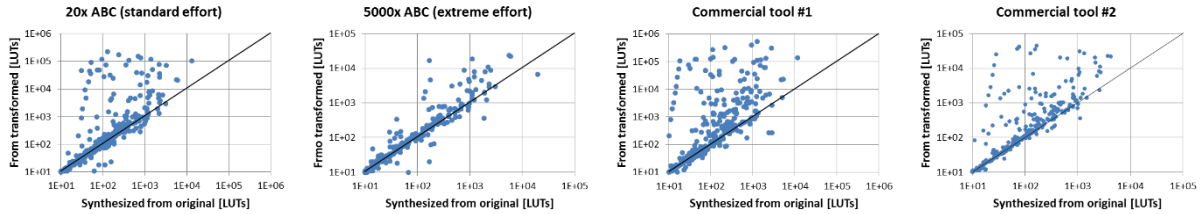

FIGURE 3. Comparison of sizes of synthesized original and transformed circuits – BDD transformation

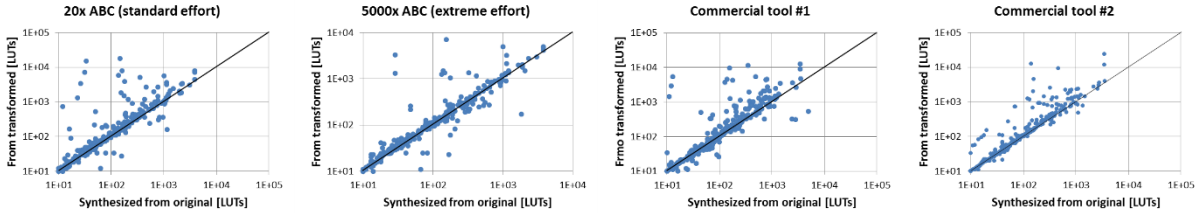

FIGURE 4. Comparison of sizes of synthesized original and transformed circuits – "*collapse*" + "*tech_decomp*"

LETHAL CASES IN STANDARD BENCHMARK SETS

We have shown that in many cases both enlarging and destroying the original benchmark circuit structure leads to an unacceptable synthesis result size increase. However, circuits probably processed in the same way are rather frequently present in standard benchmark sets too. For example, circuits in the IWLS'93 benchmark set [9] are decomposed into a network of inverters and AND and OR gates, probably in a very inefficient way (unknown to the authors, since it is not documented). Collapsing the network into a SOP will significantly help in many cases (in 40%); the observed maximum size decrease after collapsing and synthesis was 20-fold.

Also, XOR intensive circuits ('*alu4*', *'t481'*, '*xor5'*) are often presented in their collapsed form in standard benchmark sets [10, 14]. This yields a significant size blowup, when synthesis tools are not able to discover the XORs.

NON-LETHAL TRANSFORMATIONS AND REMEDIAL ACTIONS

We have shown that some circuit transformations, like SOP collapsing, BDD transformation, or "blind" decomposition, increase the circuit size and the synthesis result size is accordingly increased too, which is not welcome. There are also transformations that increase the circuit size too, but synthesis is able to efficiently cope with them. A trivial case would be addition of a chain of inverters; we can increase the circuit size arbitrarily, however, the inverter chains will be efficiently eliminated by any synthesis.

Another, less trivial case is shared logic replication [18]. The circuit size can be significantly increased, up to a complete elimination of any branching. We have observed that ABC is completely immune to this transformation, the two tested commercial tools are sensitive only insignificantly [18].

In general, a designer should avoid actions that increase the circuit size, otherwise a result size increase should be expected. Next, a proper decomposition could mitigate the effects of some transformations [19]. For example, the BDS decomposition tool [20] is able to efficiently re-discover XOR gates eliminated by network collapsing.

CONCLUSIONS

Several simple circuit transformations increasing the circuit size were presented. We have shown that the source circuit size increase induces proportional synthesized result size increase, even in case of an extreme-effort synthesis. Of course, this applies to some transformations and some circuits only; there are transformations that are not lethal to synthesis, there are circuits that are immune to some transformations, and also there are special synthesis processes that can eliminate the effects of transformations for some circuits. But still, there remains an open field and a big challenge for research; the ultimate logic synthesis process should not exhibit any structural bias and produce near-optimum results for arbitrary circuit descriptions. This is definitely not the case of contemporary academic and commercial tools.

ACKNOWLEDGEMENT

Computational resources were provided by the MetaCentrum under the program LM2010005 and the CERIT-SC under the program Centre CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, Reg. no. CZ.1.05/3.2.00/08.0144.

REFERENCES

1. G. D. Hachtel and F. Somenzi, "*Logic Synthesis and Verification Algorithms*," Boston, MA, Kluwer Academic Publishers, 1996, 564 p.
2. 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.
3. K. Karplus, "Using If-Then-Else DAG's for Multi-Level Logic Minimization," Univ. California, Santa Cruz, UCSC-CRL-88-29, Nov. 1988, p. 21.
4. Berkeley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and Verification" [Online]. Available: http://www.eecs.berkeley.edu/alanmi/abc/.
5. J. Cong and K. Minkovich, "Optimality study of logic synthesis for LUT-based FPGAs", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 26, Issue 2, Feb. 2007, pp. 230–239.
6. P. Fišer and J. Schmidt, "A Difficult Example or a Badly Represented One?" in Proc. of 10th International Workshop on Boolean Problems (IWSBP), Freiberg, Germany, September 19–21, 2012, pp. 115–122.
7. P. Fišer and J. Schmidt, "Small but Nasty Logic Synthesis Examples," in Proc. of 8th Int. Workshop on Boolean Problems (IWSBP), Freiberg (Germany), September 18–19, 2008, pp. 183–190.
8. P. Kubalík, P. Fišer, and H. Kubátová, "Fault Tolerant System Design Method Based on Self-Checking Circuits," in Proc. of 12th Intl. On-Line Testing Symposium (IOLTS), Como (Italy), July 10–12, 2006, pp. 185–186.
9. K. McElvain, "IWLS93 Benchmark Set: Version 4.0," Mentor Graphics, May 15, 1993, p. 6.
10. R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation," in *IEEE Transactions on Computers*, Vol. 35, No. 8, August 1986, pp. 677–691.
11. S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide," Technical Report 1991-IWLS-UG-Saeyang, MCNC, Research Triangle Park, NC, January 1991, p. 45.
12. 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.
13. 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.
14. S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Technical Report 1991-IWLS-UG-Saeyang, MCNC.
15. C. Albrecht, "IWLS 2005 Benchmarks," published at 2005 International Workshop on Logic Synthesis.
16. F. Somenzi, "CUDD: CU Decision Diagram Package Release 2.4.1," University of Colorado at Boulder, http://vlsi.colorado.edu/~fabio/CUDD [Online].
17. P. Fišer and J. Schmidt, "On Using Permutation of Variables to Improve the Iterative Power of Resynthesis," in Proc. of 10th Int. Workshop on Boolean Problems (IWSBP), Freiberg (Germany), 2012, pp. 107–114.
18. P. Fišer and J. Schmidt, "New Ways of Generating Large Realistic Benchmarks for Testing Synthesis Tools," in Proc. of 9th Int. Workshop on Boolean Problems (IWSBP), Freiberg, Germany, 2010, pp. 157–164.
19. P. Fišer and J. Schmidt, "The Case for a Balanced Decomposition Process," in Proc. of 12th Euromicro Conference on Digital Systems Design (DSD), Patras (Greece), August 27–29, 2009, pp. 601–604.
20. C. Yang and M. Ciesielski, "BDS: A BDD-Based Logic Optimization System," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 21, No. 7, 2002, pp. 866–876.