A Difficult Example Or a Badly Represented One?

Petr Fišer, Jan Schmidt
Czech Technical University in Prague
email: fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

Abstract

The causal connection between input circuit representation and the quality of the synthesis result is investigated, with special attention to the LEKU examples of Cong and Minkovich. It is shown that transformations totally obscuring the original circuit structure can enlarge the input to a great degree, that the LEKU circuits are nothing special in this respect, and that contemporary tools invariably produce poor results. On the other hand, such descriptions do not occur in practice.

1 Introduction

Cong and Minkovich [1] published examples targeted at the mapping phase of logical synthesis, for which their optimal implementations are known – the LEKO examples. These examples have description variants, which can be used to test the entire logic synthesis. Only the upper bound of optimal implementation size is known (LEKU examples), they have a two-level unbalanced (LEKU-CD) structure and proved to be very hard for any synthesis process. Typical resulting circuits are more than 400-times larger than expected. There are also balanced versions (LEKU-CB), which are less hard (up to 5-times larger results).

The authors proved that structural parameters of the circuit are similar to practical circuits. To our knowledge, the cause of the observed (and rather alarming) synthesis results is not known, nor is the linkage between them and practical performance of synthesis tools.

We proved [2], that the examples can be solved with reasonable success by the "old textbook" synthesis based on two-level minimization and decomposition. We therefore conjectured that the circuit structure misleads processes that preserve input structure and optimize it gradually.

This leads immediately to the notion of "good" and "bad" circuit description, which we are going to discuss in this contribution. If inputs to synthesis tools can be in nature similar to LEKU-CD, then there are huge opportunities to improve performance. Are the currently used benchmarks really difficult, or only "badly" described?

We created a process which yielded "best" and "worst" descriptions, and observed where the actual descriptions lie. This is, of course, subject to definitions of "good" and "bad", and to our ability to find such descriptions.

We briefly recapitulate the process (data flow) of Cong and Minkovich first. Then we discuss the process which we derived from that flow and used in our experiments. Finally we present and interpret our observations resulting from the experiments.

2 LEKU Circuits of Cong and Minkovich

The set of examples is defined using a relatively small circuit described as a Boolean network with two-input nodes, a replication algorithm which can produce circuits of unlimited size, and a proof of optimum mapping for the replicated circuit. The replication ensures that there is a path from each input to each output. When used for synthesis evaluation, the circuit synthesized from an altered description is compared with the proved optimum mapping. As the synthesis can produce a better result than mere mapping, the optimum mapping is only an upper bound of the result size, and hence the name – Logic synthesis Examples with Known Upper bounds (LEKU).

To produce the altered synthesis input, the original description is collapsed into Sum of Products (SOP), which is then converted into structural description by applying the SIS command

tech_decomp [20] or the ABC command *balance* [21] (Figure 1). A network of two-input gates is obtained as a result.

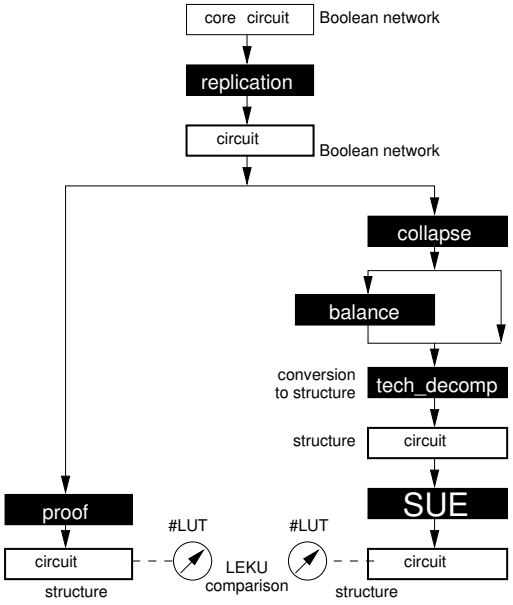

Figure 1: Circuit construction and data flow of Cong and Minkovich for a SUE (Synthesis Under Evaluation)

3 Metrics and processes

The data flow of Cong and Minkovich tests a synthesis process. As we are going to study the influence of circuit description on synthesis performance, we have to change the flow while staying as close to the original procedure as possible. Firstly, we have to work with a number of practical circuits, that is, with public benchmark sets. Secondly, we have no upper bounds for the results. Thirdly, it is not feasible to measure every existing synthesis tool.

Therefore, we have chosen a single synthesis tool to represent state-of-the-art and have prepared the benchmarks to mimic random logic synthesis in a complex design flow. Then we compared circuits that resulted from synthesis of the original and altered description (Figure 2).

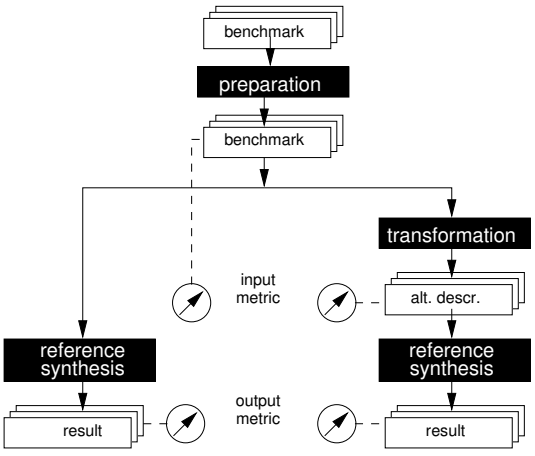

Figure 2: Data flow to study the influence of an alternative circuit description on the synthesis result

3.1 Reference synthesis tool

For this study, we need tools which are scalable, to be coherent with practice. We performed pilot experiments with some industrial and academic tools, which revealed their strong and week spots. The resistance against input structure alterations was the best with ABC [21]. We used the sequence *dch, if, lutpack* of ABC commands to perform the tests, as it was the recommended sequence for synthesis and LUT mapping at that time. We will refer to this sequence simply as *dch_if_lutpack* in the following text.

3.2 Benchmarks preparation and selection

Working with a tool intended to study synthesis of random logic, such as ABC, could cause loss of relevance. Industrial tools comprise [3], besides algorithms for random logic synthesis, a number of specialized algorithms detecting and generating a certain class of circuits, such as state machines or arithmetic (Fig. 3). Hence, we had to exclude benchmarks which normally would not be processed by a random logic synthesizer.

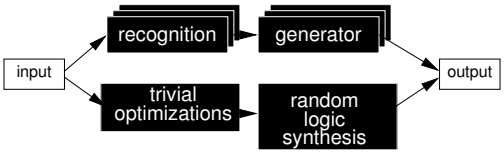

Figure 3: Possible top-level architecture of a synthesis tool

Synthesis of random logic begins with constant propagation and certain trivial optimizations, such as the removal of excessive inverters. We chose to model this step by the ABC *sweep* command [21]. Such a process can of course substantially differ from tool to tool. Even the corresponding command in SIS [20] gives results of different size, because of a capability of using 'inverted nodes' in ABC.

We used benchmark sets from the following sources, with the total of 490 circuits:

- ISCAS'85 [5]
- ISCAS'89 [6]
- MCNC [4]
- LGSynth'91 [7]
- IWLS'93 [8]
- ITC'99 [9]
- Industrial designs from Altera, available at OpenCores [10], converted by Alan Mishchenko
- Mentor Graphics benchmarks, converted by Alan Mishchenko
- Illinois test generation benchmarks [11]

3.3 Transformations

We reproduced the transformation used by Cong and Minkovich as accurately as possible, using the ABC commands *collapse* and *tech_decomp*. The latter command prevents ABC from discovering that the input is a SOP bearing no useful structure, which we verified.

A transformation effective in our experiments must obscure the original structure completely. The only such transformation we know of, besides collapsing to SOP, is to build an ordered BDD [12], [13] and to construct structural description out of it (a network of multiplexers). By forcing random BDD variable ordering, we obtain multiple descriptions of a single circuit. We have used the CUDD package [14] for this purpose.

To measure minimum description size, we used a variety of processes. In some cases BDS [15] followed by ABC *dch_if_lutpack* was successful. In others, repetitive use of different synthesis steps brought the best result.

3.4 Metrics

To be compatible with Cong and Minkovich, we synthesized all circuits to 4-input look-up tables (LUTs), even though more recent devices have different programmable blocks. The output

metric in Figure 2 is therefore the number of LUTs.

Cong and Minkovich concentrated on circuit size, not circuit timing. We kept this focus, and not only for compatibility reasons. If some synthesis process produces a circuit orders of magnitude larger than it should be, then its timing is quite uninteresting. Moreover, the circuits optimized for speed and for area usually differ much less than the observed variations in our experiments.

Unlike Cong and Minkovich, we were also interested in the change of circuit size caused by transformation of circuit description. Here we must avoid any metric which would require synthesis steps (such as LUT number or SOP size). We used the number of literals as the input metric in Fig. 2.

4 Changes in result size versus changes in input size

We studied how *dch_if_lutpack* responds to alteration of its input. For each of the 490 benchmarks, we measured *result enlargement* as the size of the circuit synthesized from the alternative input relative to the size of the circuit synthesized from the original circuit. Similarly, we measured *input enlargement* as the size of the altered input to the size of the original input.

Figure 4 shows result enlargement as a function of input enlargement when *collapse* and *tech_decomp* was used as the altering transformation. The diagonal line has the slope 1.

Three regions can be observed. In the first region on the left, the transformation *shrinks* the circuit. *dch_if_lutpack* often does not use the advantage, and returns a circuit similar in size to that before the transformation, i. e. larger.

In the second region, which covers input enlargement from 1 to circa 8, *dch_if_lutpack* can mostly compensate for the enlargement. Again a circuit similar in size to the original one is resulted, which is acceptable behavior.

In the third region above 8, *dch_if_lutpack* gives up and the size of its output follows the size of its input, in this case reduced by a constant factor. The correlation of the dependency over all three regions is 0.988.

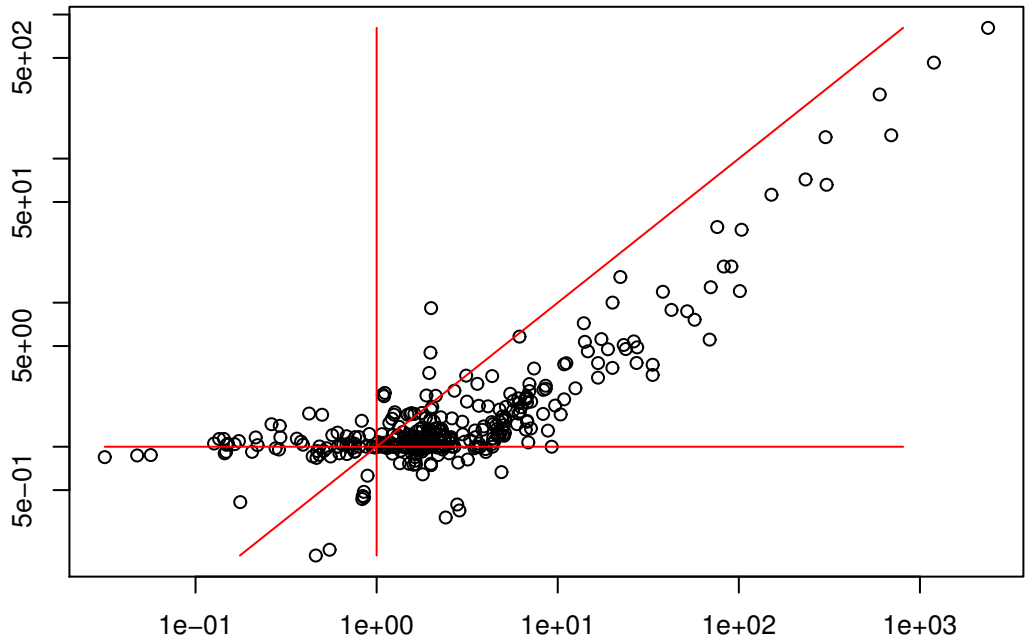

Figure 4: Result enlargement as a function of input enlargement, *collapse* and *tech_decomp* as transformation

Figure 5 shows the same function, in this case with 20 randomly ordered BDDs as transformations. Similar phenomena can be observed. The correlation is 0.857 over all transformations,

and 0.942 when the ordering is not random but taken over from the input. In both graphs we

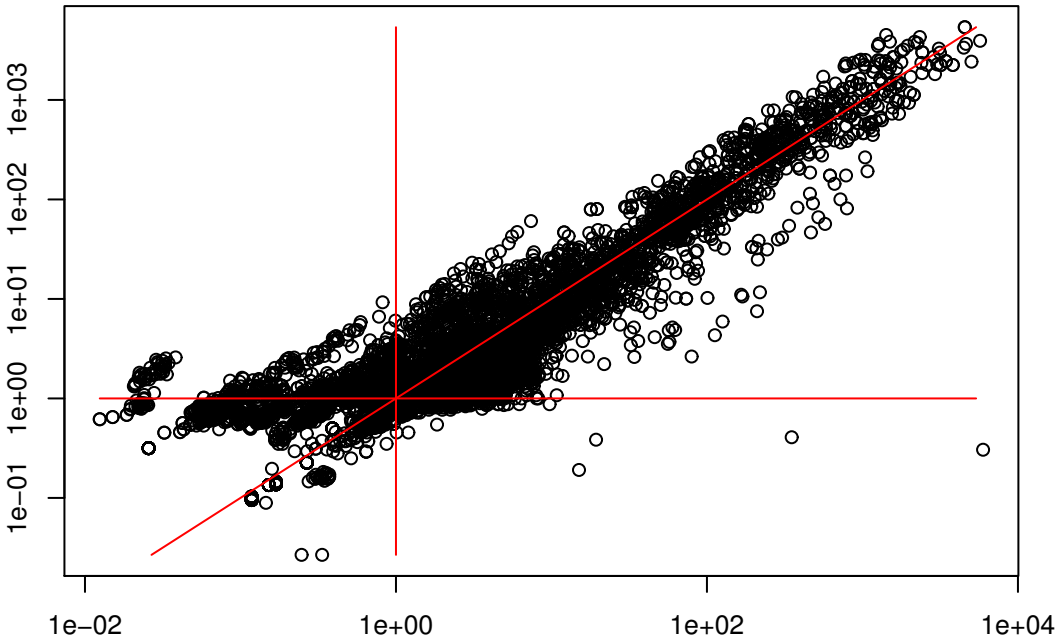

Figure 5: Result enlargement as a function of input enlargement, randomly ordered BDDs as transformations

can observe that where the transformation succeeds in enlarging the circuit representation, the size of output from *dch_if_lutpack* also grows proportionally.

5 Input size span and the original input position

Having observed what a large circuit description can cause, we can answer the question whether practical circuit descriptions are "good" or "bad".

First, we are going to find the smallest and the largest obtainable input. The minimum representation is simply minimum over results of all synthesis procedures tested, most notably including the "old textbook" BDS [15], which also performs XOR decomposition. The maximum is a 'reasonable' maximum; one can increase the representation size endlessly by introducing inverter chains etc. Here it is the maximum achieved by transformations obscuring the original structure, using either collapsing to SOP or BDD construction.

Then we can tell whether the original description of any benchmark is closer to the minimum or maximum. The result is in Figure 6. Each horizontal line represents one circuit. The left end represents the size of its smallest description, the right end the largest. The circuits are ordered by the geometric average of the minimum and maximum.

The circular marks in the graph give the relative result enlargement caused by the largest description. For some circuits, the mark is missing, as *dch_if_lutpack* failed to produce a result within 2 days of computation and 1GB of memory.

Again three regions can be observed. In the first region at the bottom, the description is so poor that merely collapsing the circuit or producing a BDD helps greatly. As can be seen in Figures 4 and 5, *dch_if_lutpack* mostly does not utilize the advantage.

In the middle region, the original description size is roughly centered between minimum and maximum. Obscuring the circuit structure leads to substantial result enlargement in some cases, even when the input enlargement is not high. In most cases however, the tool can manage to achieve 'normal' output.

In the third, uppermost region, the transformation managed to enlarge the input to a great degree – up to four orders of magnitude. This is the region where LEKU-CD resides. Scalable

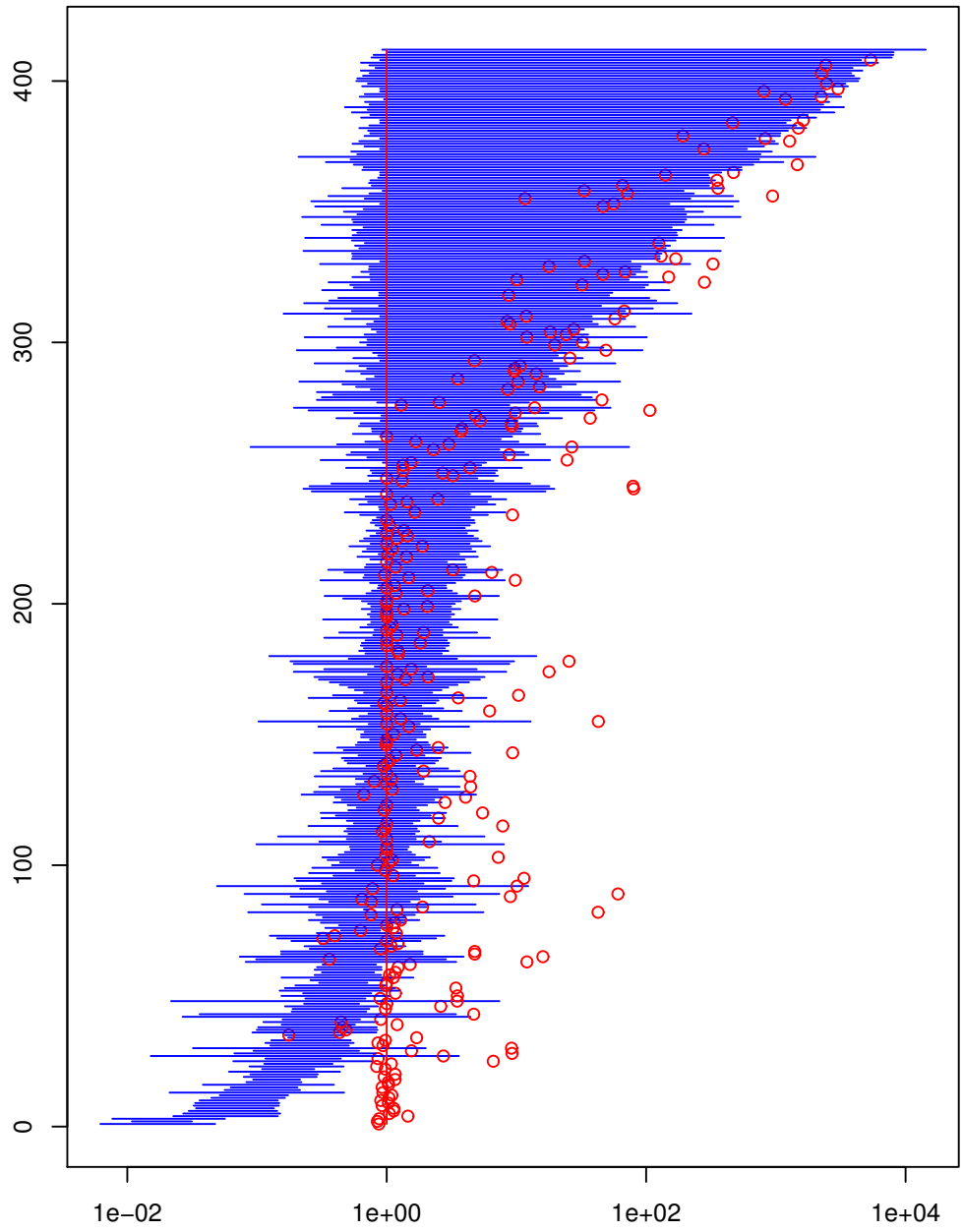

Figure 6: Input size spans and *dch_if_lutpack* performance on the largest inputs

tools such as *dch_if_lutpack* give invariably poor results. Notice however, that the original descriptions are rather close to minimum. This means that there are no benchmarks similar in nature to LEKU-CD, and that the ability to produce a good result from a very bad description has limited practical importance. In turn this nevertheless means that smooth operation of contemporary synthesis does depend on the designer, who is supposed to enter the circuit with reasonable structure.

A special case of structure transformation shall be mentioned here. A two-level representation of a XOR-intensive circuit can grow rapidly with the size of the input. The gap is exponential in the worst case. Medium size circuits, for example *xor5* and *t481* [8], can stay in the middle region of Figure 6. Many tools cannot handle such circuits well [19] and produce poor results, contributing to the observed character of the middle region.

One example of practical circuits of this class are parity predictors [17][18], which we analyzed in [16]. A parity predictor is composed of a combinational circuit and a parity tree at the output. Here the designers *deliberately threw away* the original structure, in the hope of obtaining smaller

circuits. This is, as it is now clear, not what the synthesis tools are made for. While the results were small in many cases, there were cases of surprising result enlargement (e.g., from 11 to 298 4-LUTs for the *alu1* circuit [7]).

The techniques developed for this study can be used for other purposes, too. The set of transformations forcing a particular tool to produce poor results can tell much about the algorithms used, to the extent that such experiments could be considered reverse engineering. Also, to compare tools on "bad" descriptions is tempting; we shall nevertheless keep in mind that their practical relevance is limited. In this study, we limited our focus to put the examples of Cong and Minkovich into a broader perspective.

6 Conclusions

The question in the title, "a difficult example or a badly represented one?" can be answered "mostly the former". The phenomena observed with LEKU examples of Cong and Minkovich can occur with other publicly available benchmarks. They have, however, limited significance for practical application, as circuit representations of equal nature do not occur. The quality of synthesized circuits depends on the designer's ability to structure the circuit well. Where the circuit structure has to be neglected for any reason, contemporary tools cannot be relied on. If such a circuit is within scalability limits of the "old textbook" synthesis flow based on decomposition and capable of XOR handling, then substantially better results could be achieved.

Acknowledgment

We are thankful to dr. Minkovich for the LEKU circuits, and to dr. Mishchenko for converting many benchmarks to BLIF.

References

[1] 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 2007, 26 (2), pp. 230–239. Postprint available free at: http://repositories.cdlib.org/postprints/2376

[2] P. Fišer and J. Schmidt, "The Observed Role of Structure in Logic Synthesis Examples", Proceedings of the International Workshop on Logic and Synthesis 2009, Berkeley, CA, USA, pp. 210–213.

[3] Oral tradition in the EDA community.

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

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

[6] F. Brglez, D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", Proceedings of the International Symposium of Circuits and Systems, 1989, pp. 1929–1934.

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

[8] "IWLS'93 Benchmark Set: Version 4.0", distributed as part of the IWLS'93 benchmark distribution.

[9] IEEE test Technology Technical Committee: "The ITC99 Benchmark Set", http://www.cerc.utexas.edu/itc99-benchmarks/bench.html

[10] Altera, inc., OpenCores, http://opencores.org/

[11] V. Chickermane, J. Lee, and J. H. Patel, "A comparative study of design for testability methods using high-level and gate-level descriptions", Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, November 1992, pp. 620–624.

[12] S. B. Akers, "Binary decision diagrams", IEEE Transactions on Computers, vol. C-27, No. 6, June 1978, pp. 509-516.

[13] R. E. Bryant, "Graph based algorithms for Boolean function manipulation", IEEE Transactions on Computers, vol. 35, No. 8, August 1986, pp. 677-691.

[14] F. Somenzi, "CUDD: CU Decision Diagram Package Release 2.4.1", University of Colorado at Boulder [Online]. Available: http://vlsi.colorado.edu/~fabio/CUDD

[15] C. Yang and M. Ciesielski, "BDS: A BDD-Based Logic Optimization System", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems Vol. 21, 2002, No. 7, pp. 866–876.

[16] P. Fišer and J. Schmidt, "Small But Nasty Logic Synthesis Examples", Proceedings of the 8th. Int. Workshop on Boolean Problems, 2008, Freiberg, pp. 183–189.

[17] P. Kubalík, P. Fišer and H. Kubátová, "Fault Tolerant System Design Method Based on Self-Checking Circuits", Proceedings of the 12th International On-Line Testing Symposium 2006 (IOLTS'06), Lake of Como, Italy, July 10-12, 2006, pp. 185–186.

[18] P. Fišer, P. Kubalík and H. Kubátová, "An Efficient Multiple-Parity Generator Design for On-Line Testing on FPGA", Proceedings of the 11th Euromicro Conference on Digital Systems Design (DSD'08), Parma (Italy), 2008, pp.96–99.

[19] P. Fišer and J. Schmidt, "The Case for a Balanced Decomposition Process", Proceedings of the of 12th EUROMICRO Conference on Digital System Design. Los Alamitos: IEEE Computer Society, 2009, pp. 601–604.

[20] E. Sentovitch, K. Singh et al., "SIS: A System for sequential circuit synthesis", Univ. California, Berkeley, Tech. Rep., UCB/ERL M92/41, May 1992.

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