XOR-Based Synthesis: Does it Pay off?

Ivo Halecek, Petr Fiser, Jan Schmidt

Czech Technical University in Prague
Faculty of Information Technology, Department of Digital Design
Prague, Czech Republic
{halecivo, fiserp, schmidt}@fit.cvut.cz

Abstract. This paper discusses the feasibility and efficiency of a native XOR-based logic synthesis. Particularly, a natively XOR-supporting rewriting algorithm is presented. It is based on a novel data structure – the XOR-And-Inverter Graph. Major problems that appeared by extending the And-Inverter Graph rewriting to support XORs are discussed. The algorithm was implemented in a logic synthesis and optimization package ABC and compared to standard And-Inverter-Graph based rewriting. Its benefits are shown by experiments.

Keywords: logic synthesis, AIG, XAIG, rewriting, XOR.

1 Introduction

Gate-level logic synthesis and optimization recently experiences renewed interest, mostly due to emergence of new physical structures implementing "non-standard" logic functions simply [1]. In a strong contrast to this, standard synthesis processes efficiently used for many decades, which are typically based on simple nodes (2-input ANDs, NORs, inverters) do not seem to be efficient anymore and new paradigms and algorithms based on them must be looked for. Examples of such novel paradigms are Majority-based [2] or XOR-Majority-based [3] approaches.

The second motivation for research of new synthesis paradigms is the observed lack of efficiency of standard synthesis, especially for XOR-intensive circuits [4, 5]. Such circuits appear ever more frequently in industrial designs, since they are parts of fault-tolerant [6] and secure [7] devices.

Since the *rewriting* logic optimization algorithm [8] represents one of the simplest ways to efficiently exploit the above-mentioned structures, it is now being used at the first place for demonstrating their feasibility and effectiveness.

The rewriting algorithm [8] takes a logic network as input (no matter what elements it is composed of), extracts small parts of it, and replaces them with functionally equivalent optimum implementations. This is repeated, until the whole network is traversed. Two issues must be resolved to accomplish this: 1) generation of the optimum replacement structures, 2) controlling the overall rewriting process.

In this extended abstract, we present an XOR-AND-Inverter Graph (XAIG) structure as an extension of a standard AND-Inverter Graph (AIG) [9, 10] enabling to represent XOR gates explicitly. Next, we introduce a rewriting algorithm based on this structure.

The two above issues become more complicated in XIAG rewriting, since in contrast to AIG, XAIG is an *heterogeneous* structure, where XOR can be decomposed to multiple AND nodes (with inverted edges provided). Moreover, there are multiple ways of doing this decomposition. Next, AND and XOR nodes may be assigned different implementation costs. Therefore, there are multiple alternatives of generating the optimum replacement structures.

The rewriting control is more complicated too. Since XOR nodes may be alternatively "dissolved" into several AND nodes in the rewriting process when looking for replacements, this issue will be discussed too.

The presented XAIG-based rewriting algorithm is compared to the standard AIG-based rewriting [8], both as stand-alone processes and when incorporated in a more complex iterative synthesis process (LUT mapping). The results indicate superiority of the XAIG-based rewriting, especially when used in combination with AIG rewriting.

The main ideas presented in this extended abstract were published in [11]. Here we just summarize the most important observations and emphasize the discussion.

2 The State-of-the-Art

The original *rewriting* process for simple AIG structures has been proposed already in 2006 [8]. Since that time, no big research on this topic existed, until recent years. New paradigms of logic synthesis emerged, accompanied by new structures. The most striking new paradigm is *majority-based* synthesis with Majority-Inverter-Graphs (MIGs) introduced [2] in 2015. This idea has been further extended to *XOR-Majority-Graphs* (XMGs) [3]. The efficiency of these structures has been demonstrated on the rewriting algorithm. The authors have shown benefits over a standard AIG-based optimization when targeted to new technologies (where the majority function can be implemented "simply) and also for standard cells or LUT mapping, where the compact majority function can be advantageously used by the mapper.

In the ABC system [12], a facility implementing Boolean networks with AND, XOR and MUX nodes together with negated edges, was recently established. The XAIG structure has also been proposed in [13, 14], but here it is used just for technology mapping purposes, not primarily for logic synthesis.

In logic optimization scenarios, XORs are rarely represented directly (except of XMGs), which could suggest certain algorithmic bias against them. To judge the role of XORs, we need a representation where ANDs and XORs would be equally "first class citizens", with balanced roles. The structure must be simple enough to permit adaptation of most base ABC algorithms, and as close as possible to AIGs for fair comparison (which unfortunately, excludes XMGs). Also, the difficulties caused by making the network heterogeneous should be kept small. Even though both XMGs and ABC AND-XOR-MUX-Inverter graphs are generalizations of XAIGs, they are not suitable for our purposes; the question we seek an answer for, is whether treating XORs and ANDs in a balanced way will improve the performance of logic synthesis. Therefore, the set of operators must be restricted to AND and XOR nodes only. After resolving this question, we may think about extending the nodes set further, if needed.

3 AIGs, XAIGs, and XAIG-Based Rewriting

And-Inverter Graph (AIG) [8, 9] is a directed acyclic graph with one or more roots, where nodes are two-input AND gates and edges represent connections between them. Edges may be inverted, meaning that the respective subgraph is negated. This can be understood as an inverter presence on the connection.

XOR-And-Inverter Graph (XAIG) is an extension (generalization) of AIG, where nodes are two-input ANDs or XORs.

A two-input XOR gate can be represented in AIG by several structurally different ways. The minimum XOR implementation consists of three AND gates, and there are two such implementations [11], [13]. Even though it is possible to construct a single XOR gate using more AND nodes, such redundant structures will not be assumed here.

The initial AIG network can be transformed to XAIG by pattern matching. This feature is already implemented in the ABC9 package, by the command '&st -m'.

Rewriting [8, 11] is a technique of replacing AIG (XAIG, in our case) subgraphs with K leaves (K-feasible cut cones) by optimum functionally equivalent structures. An example of XAIG subgraph replacement can be seen in **Fig. 1**.

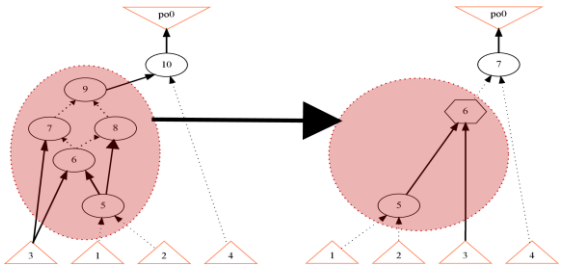

Fig. 1. XAIG based rewriting example. The oval nodes represent ANDs, the hexagon an XOR.

The algorithm goes through XAIG nodes in topological order. For each node, the subgraphs (cuts) are enumerated using an algorithm presented in [15]. For each node cut, a truth table of the function of its leaves is calculated by simulation and converted to a canonical form describing the respective *NPN equivalence* class [8, 16].

For each NPN equivalence class, there are one or more pre-computed optimum structures stored. Note that for 4-input cuts, there are 2^{16} possible functions, but there are only 222 NPN equivalent classes.

Each node cut cone is then tried for replacement by each pre-computed optimum representation. After each temporary replacement, the total network cost is calculated (see below). If at least one replacement leads to better total network cost, replacement with the best cost reduction is made permanent for the currently processed node. After that, the algorithm continues with topologically subsequent node. Possible sharing of subgraphs in the XAIG is considered [8, 9], thus, the size-optimality of the replacement structure does not guarantee the optimality of the replacement; different replacement structures may yield XAIGs of different sizes.

The cost of the network is computed as a weighted sum of nodes cost, where for each node type (AND and XOR) a cost is specified. The cost can be adjusted with respect to the expected target technology.

The optimum structures for each NPN-equivalence class are computed using a SAT-based approach presented in [17]. The XOR cost may be specified here as well, yielding different optimum implementations with different properties. Also, different numbers of optimum representations are produced depending on the XOR cost. The numbers of nodes, XORs and structures counts for different costs are summarized in **Table 1**.

Table 1. The statistic on all generated 222 NPN-equivalent 4-input replacement circuits.

AND:XOR	Nodes		XORs		Count		
	Max.	Avg.	Max.	Avg.	Max.	Avg.	Total
1:1	7	6.60	5	2.78	7,401	144.86	32,160
1:2	8	6.38	3	1.41	2,436	42.58	9,453
1:3 = no XORs	10	8.02	3	0.23	3,056	95.45	21,190

3.1 XOR transformations

The presence of XOR nodes brings additional possibilities of choice. Particularly, XOR gates, either present in the original XAIG or newly introduced by cut replacement, may or may not be "dissolved" into one of the two 3-AND structures. These alternatives are compared and the one yielding the lowest total network cost is used.

Note that the total network cost is computed considering sharing of XAIG sub-graphs; one particular replacement structure may introduce new nodes, which are structurally equivalent to other nodes already present in the XAIG. Therefore, the network cost can be computed only after the replacement is physically performed and structural sharing possibilities determined (structural hashing is done). If such sharing is found and it is found to reduce the total network cost, the XOR dissolution is made permanent. Otherwise, the XOR is collapsed back to a single node.

Apart from the basic dissolution, a duplication technique can be used. Here, when a structure of multiple nodes is to be replaced by a XOR node, some of the inner nodes may be duplicated without negatively affecting the total cost, depending on nodes cost configuration. In particular, one or both inner AND nodes of the XOR function may be duplicated to preserve inputs for nodes outside the cut. These alternatives are always tried when computing the network cost.

An example of such situation can be seen in **Fig. 2**. Here, an XOR function has been found, however one of the inner nodes has an edge leading outside the cut. This cut can therefore be replaced by a XOR node, but the inner node has to be duplicated to preserve this output leading outside of the cut ('network b'). Other option is to keep the representation of the function by three AND nodes, as it is in 'network a'. The final decision will depend on the AND:XOR cost ratio.

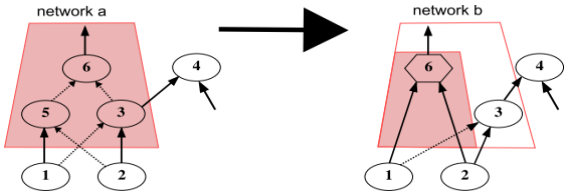

Fig. 2. Duplication of AND nodes after cut replacement

4 Experimental Results

As a comparison of the AIG-based synthesis with the XAIG-based synthesis, we have run both rewriting algorithms over a set of more than 700 circuits obtained as a mix of different benchmarks: LGSynth'91 [18], IWLS'93 [19], ISCAS'85 [20], ISCAS'89 [21], ITC'99 [22], EPFL [23], IWLS 2005 [24], and LEKO/LEKU benchmarks [25] – all available from [26].

4.1 Comparison of AIG and XAIG-Based Rewriting

A summary comparison of the original AIG rewriting (*rewrite*) to XAIG rewriting (*&rewrite*) with different AND:XOR cost settings can be seen in **Table 2**. For each XOR cost setting used with XAIG rewriting, optimum replacement structures with the same cost setting has been generated. The counts of resulting nodes (total and XOR nodes) and circuit levels were measured. Only sums of the respective values over all 700 circuits are shown here. For details on individual circuits see [11].

Table 2. Comparison of the AIG-based rewriting (*rewrite*) and XAIG-based rewriting (*&rewrite*) with different AND:XOR cost settings

Process	Nodes	XORs	Cost	Levels
rewrite	606,851	18,772	635,009	**12,877**
&rewrite, AND:XOR cost 1:1	**598,802**	**25,510**	**598,802**	13,035
&rewrite, AND:XOR cost 1:2	603,772	17,195	620,967	13,096
&rewrite, AND:XOR cost 1:3	613,691	10,438	634,567	13,281

The results show that with increasing cost of a XOR node, the algorithm prefers AND nodes over XORs and the total number of nodes naturally increases. Although these results may seem to be quite obvious, they fully expose the "strength" of having native XOR nodes; when XORs are allowed in addition to ANDs, the circuits can be implemented using fewer gates.

The "Cost" is computed as respectively weighted sum of nodes count. The AND:XOR cost for the standard AIG-based rewriting was set to 2:5. We see that the best results have been obtained for AND:XOR cost 1:1, even in terms of the total cost. Surprisingly enough, the original AIG rewriting yielded results with less levels (delay).

4.2 Overall synthesis process – FPGA mapping

To demonstrate the influence of XAIG rewriting on the overall synthesis process we compared both algorithm variants by the number of LUTs and levels after iterated rewriting and mapping to FPGA (ABC script '*rewrite; balance; if; mfs*' iterated 20-times, for XAIG-based rewriting, the '*&rewrite'* command was used instead of '*rewrite'*). In order to see the role of the XOR cost in the LUT mapping process (where, indeed, different costs hardly make sense), we have performed this experiment using three different XOR costs as well.

The results are shown in **Table 3**. Only 13 largest circuits are shown there, with summary values for all 700 circuits shown in the last row. We can see that the XOR cost equal to 1 produced the best results again, in terms of the area (LUTs count). Particularly, better results than the AIG rewriting were obtained in 290 cases (43%), worse results in 146 cases (21%), out of 682. The original AIG-based rewriting procedure produced the best results only rarely. This result just confirms the conjecture that XAIG rewriting with the XOR cost set equal to the AND cost yields best results, emphasizing the importance of XOR nodes.

Note that in this experiment, the influence of algorithmic noise [27, 28] has been suppressed by averaging results from at least 40 runs with randomly permuted inputs and outputs. Therefore, even though the improvement is negligible, it is *systematic*. In other words, generally we cannot lose when *&rewrite* is used instead of *&rewrite*.

Table 3. Comparison of the AIG-based rewriting (*rewrite*) and XAIG-based rewriting (*&rewrite*) with different AND:XOR cost settings – LUT mapping

	rewrite		*&rewrite* 1:1		*&rewrite* 1:2		*&rewrite* 1:3	
name	LUTs	Lvl.	LUTs	Lvl.	LUTs	Lvl.	LUTs	Lvl.
arbiter [23]	4,053	30	4,053	30	4,053	30	4,053	30
apex2 [19]	1,696	7	1,690	7	1,679	7	1,679	7
bigkey [19]	1,695	3	1,789	3	1,898	3	1,901	3
too_large [19]	1,475	8	1,504	8	1,377	9	1,377	9
mainpla [18]	1,419	10	1,394	10	1,403	10	1,391	10
dsip [19]	1,360	3	909	3	908	3	908	3
misex3 [19]	1,358	6	1,296	7	1,285	7	1,285	7
bar [23]	1,349	6	1,349	6	1,349	6	1,349	6
des [19]	1,347	6	1,289	7	1,349	6	1,331	6
xparc [18]	1,316	11	1,319	11	1,319	11	1,330	11
spi [24]	1,252	10	1,237	10	1,246	10	1,254	10
wb_dma [24]	1,246	8	1,230	11	1,231	11	1,233	11
apex4 [19]	1,137	6	1,083	7	1,090	7	1,100	7
Total	**126,258**	**3,619**	**124,066**	**3,700**	**124,789**	**3,673**	**125,016**	**3,676**

4.3 Combined synthesis procedure

From the above experiments it is apparent that sometimes a better solution was found by the XAIG-based rewriting, sometimes worse, and this cannot be attributed to the algorithmic noise (since it has been eliminated). Thus, generally speaking, XAIG-based rewriting may bring benefits for some circuits, while for some circuits it does not help.

When keeping this in mind, we can suggest an improved synthesis procedure, that always produces equal or better results than the original AIG rewriting based one: to run both synthesis procedures simultaneously (e.g., by employing two CPU cores) and pick the better result. However, we will show that even this is not necessary; half of the number of iterations is usually sufficient to obtain better results in most of cases. This can be explained by the fact, that the iterative process (*rewrite*-based or *&rewrite*-based) quickly gets stuck in a local optimum and does not further improve much with later iterations. The results in **Table 4**, for 13 biggest circuits. Here results of the *rewrite*-based process run iteratively 40-times are compared to 20 iterations of both with the better result taken (a choice is made). Thus, the total run times of both complete processes were approximately equal. Since the 1:1 AND:XOR cost ratio setting led to best results in the previous experiment, only this option was used here. The initial theory was confirmed — the combined process gave better results in most of cases. Altogether, there were only 10 circuits for which the combined process gave slightly worse results. These were mostly the biggest circuits that needed more iterations to converge. These circuits are seen in the upper part of the table.

Table 4. Comparison of the AIG-based rewriting (*rewrite*), XAIG-based rewriting (*&rewrite*) and the combined procedure

	40x rewrite		*40x &rewrite* 1:1		*combined*	
name	LUTs	Lvl.	LUTs	Lvl.	LUTs	Lvl.
arbiter [23]	4,053	30	4,053	30	4,053	30
apex2 [19]	1,607	7	1,621	7	1,690	7
bigkey [19]	1,583	3	1,789	3	1,695	3
too_large [19]	1,371	8	1,397	8	1,475	8
mainpla [18]	1,419	10	1,394	10	1,394	10
dsip [19]	1,360	3	909	3	909	3
misex3 [19]	1,252	6	1,206	7	1,296	6
bar [23]	1,349	6	1,349	6	1,349	6
des [19]	1,334	6	1,289	7	1,289	6
xparc [18]	1,316	11	1,319	11	1,316	11
spi [24]	1,252	10	1,237	10	1,237	10
wb_dma [24]	1,246	8	1,230	11	1,230	8
apex4 [19]	1,137	6	1,083	7	1,083	6
Total	**112,080**	**3,357**	**110,379**	**3,432**	**110,108**	**3,335**

5 Conclusions and Discussion

A novel circuit representation structure – the XOR-AND-Inverter Graph (XAIG) has been proposed in this paper, together with a rewriting algorithm based on this representation.

The algorithm was implemented in the framework of logic synthesis and optimization tool ABC. The XAIG-based rewriting algorithm was compared to the original AIG-based rewriting already implemented in ABC. The results indicate that the new algorithm is stronger in XOR identification and in reducing the number of nodes.

The impact of the XAIG-based rewriting process to a complete synthesis, particularly FPGA LUT mapping, was studied. When compared with the standard AIG-based rewriting process, better results were obtained in most cases.

The XOR nodes cost can be freely adjusted both in the optimum replacement structures generation and in the rewriting algorithm. However, we have found experimentally that the AND:XOR ratio 1:1 option produces best results universally, as the cost of the results improves with the amount of XORs in replacements. This in part confirms our conjecture about the XOR importance. Moreover, as shown in [11], it permits to use the replacements generated for the AND:XOR ratio 1:1 universally, even for standard cells mapping, where the XOR cost is higher than the AND cost. This phenomenon can be explained by the fact that the mapping process may benefit from the XOR presence, no matter what the XOR cost in the target library is. Moreover, the number of optimum replacement structures is for the AND:XOR ratio significantly higher than for other ratios. Thus, this allows more freedom in the rewriting process (see **Table 1**).

AIGs are, as mentioned earlier, a logically complete system. Nodes of any newly introduced type can be therefore replaced by subgraphs with AND nodes only. In our case, there are two distinct 3-AND subgraphs replacing an XOR. This has several consequences. The two representations can be interpreted in the sense an XOR in XAIG implicitly representing two different AND-based structures. This is especially important for the number of replacement structures produced, and subsequently for the rewriting run-time. Particularly, if all AND-XOR structures were explicitly generated as replacement circuits, their number will be exponential with the number of XORs ($2^{\#XORs}$). However, by representing XOR *implicitly* by one node, the rewriting time complexity is *linear* with their number, as XORs are processed one-by-one, without any dependence of previously made decisions on their dissolving.

However, there is one drawback involved. The rewriting algorithms are based on cut generation [15], which is a purely structural procedure. When "macro" XOR nodes are introduced, less cuts can be constructed and considered for replacement, leading to possibly worse results, as shown in the experimental section.

Summarized, the newly proposed XAIG-based rewriting algorithm offers a possibility of discovering new XOR structures in a network, compared to the state-of-the-art. These XORs may be utilized in further network processing algorithms. Discovery of new XORs also yields better synthesis results in a number of cases, mostly in XOR-

intensive circuits, while for the rest of circuits, comparable results are obtained. A combined procedure with superior results was demonstrated. Therefore, we can conclude that efficient and balanced handling with XORs in synthesis is useful for improving synthesis results.

Acknowledgments

This research has been partially supported by the grant GA16-05179S of the Czech Grant Agency, "Fault-Tolerant and Attack-Resistant Architectures Based on Programmable Devices: Research of Interplay and Common Features" (2016-2018) and by the grant SGS17/213/OHK3/3T/18.

Computational resources were provided by the CESNET LM2015042 and the CERIT Scientific Cloud LM2015085, provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures".

The authors acknowledge the support of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

Last, but not the least, numerous thanks to Alan Mishchenko, for his valuable comments and discussions with him.

References

1. Amaru, L. et al.: New Logic Synthesis as Nanotechnology Enabler. Proceedings of the IEEE, vol. 103, no. 11, Nov. 2015, pp. 2168-2195 (2015).
2. Amaru, L., Gaillardon, P.-E., De Micheli, G.: Boolean logic optimization in majority-inverter graphs. In: 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), pp. 1–6 (2015).
3. Haaswijk, W., Soeken, M., Amaru, L., Gaillardon, P.-E., De Micheli, G., A Novel Basis for Logic Rewriting. Tech. rep. Integrated Systems Laboratory, EPFL, Lausanne, Switzerland (2017).
4. Fišer, P. Schmidt, J.: Small but Nasty Logic Synthesis Examples. In: 8th International Workshop on Boolean Problems (IWSBP), pp. 183–189 (2008).
5. Fišer, P. Schmidt, J.: The Observed Role of Structure in Logic Synthesis Examples. In: Proc. of the International Workshop on Logic and Synthesis (IWLS), pp.210–213 (2009).
6. Pradhan, D. K.: Fault-tolerant computer system design, Prentice Hall, New Jersey, 550 p. (1995).
7. Announcing the Advanced Encryption Standard (AES). Federal Information Processing Standards Publication no. 19 (Nov. 2001).
8. Brayton, R.K., Mishchenko, A., Chatterjee, S.: DAG-aware AIG rewriting: a fresh look at combinational logic synthesis. In: 43rd ACM/IEEE Design Automation Conference, ACM, pp. 532–535 (2006).
9. Kuehlmann, A. Paruthi, V., Krohm, F., Ganai, M.: Robust Boolean reasoning for equivalence checking and functional property verification, IEEE Trans. Computer Aided Design of Integrated Circuits and Systems, 21 (12), pp. 1377–1394 (2001).
10. Biere, A.: AIGER, http://fmv.jku.at/aiger/ (2007).
11. Háleček, I., Fišer, P., Schmidt, J.: Towards AND/XOR Balanced Synthesis: Logic Circuits Rewriting with XOR. Microelectronics Reliability, Elsevier, vol. 81, pp. 274-286 (2018).

12. Mishchenko, A. et al.: ABC: a system for sequential synthesis and verification, http://www.eecs.berkeley.edu/~alanmi/abc (2012).
13. J. M. Matos et al., “Mapping circuits with simple cells from xor-and-inverter graphs,” in Proc. of Int’l Workshop on Logic and Synthesis (2015).
14. J. M. Matos, J. Carrabina, A. I. Reis, "Efficiently Mapping VLSI Circuits with Simple Cells", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2018).
15. Mishchenko, A. et al.: Technology mapping with Boolean matching, supergates and choices. ERL Technical Report, EECS Dept., UC Berkeley, Tech. Rep., 03 (2005).
16. Huang, Z., Wang, L., Nasikovskiy, Y., Mishchenko, A.: Fast Boolean matching based on NPN classification, International Conference on Field-Programmable Technology (FPT), pp. 310–313 (2013).
17. Háleček, I., Fišer, P., Schmidt, J.: SAT-Based Generation of Optimum Function Implementations with XOR Gates. In: 20th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, pp. 163–170 (2017).
18. Yang, S.: Logic Synthesis and Optimization Benchmarks User Guide: Version 3.0, MCNC Technical Report, Jan 1991.
19. McElvain, K.: IWLS’93 Benchmark Set: Version 4.0, Tech. rep. (May 1993).
20. Brglez, F., Fujiwara, H.: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In: IEEE International Symposium Circuits and Systems (ISCAS’85), pp. 677–692 (1985).
21. Brglez, F. Bryan, D. Kozminski, K.: Combinational profiles of sequential benchmark circuits, IEEE International Symposium on Circuits and Systems, vol.3, pp. 1929–1934 (1989).
22. Corno, F. Reorda, M., Squillero, G.: RT-level ITC’99 benchmarks and first ATPG results, IEEE Des. Test Comput. 17 (3) pp. 44–53 (2000).
23. Amaru, L.: The EPFL Combinational Benchmark Suite, Tech. rep. Integrated Systems Laboratory, EPFL, Lausanne, Switzerland (2016).
24. Albrecht, C., IWLS 2005 Benchmarks, Tech. rep. (Jun. 2005).
25. Cong, J. Minkovich, K.: Optimality Study of Logic Synthesis for LUT-Based FPGAs, 14th International ACM Symposium on Field-Programmable Gate Arrays, pp. 33–40 (2006).
26. Fišer, P., Schmidt, J.: A Comprehensive Set of Logic Synthesis and Optimization Examples, In: 12th International Workshop on Boolean Problems (IWSBP), pp. 151–158 (2016). http://ddd.fit.cvut.cz/prj/Benchmarks/
27. Schmidt, J., Fišer, P., Balcárek, J.: On Robustness of EDA Tools, Euromicro Conference on Digital System Design Architectures, Methods and Tools, pp. 427–434 (2014).
28. Shum, W., Anderson, H.J., Analyzing and predicting the impact of CAD algorithm noise on FPGA speed performance and power, International ACM Symposium on Field-Programmable Gate Arrays, pp. 107–110 (2012).