On Logic Synthesis of Conventionally Hard to Synthesize Circuits Using Genetic Programming

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

Zdeněk Vašíček, Lukáš Sekanina
Faculty of Information Technology
Brno University of Technology
Brno, Czech Republic
vasicek@fit.vutbr.cz, sekanina@fit.vutbr.cz

Abstract—**Recently, it has been shown that synthesis of some circuits is quite difficult for conventional methods. In this paper we present a method of minimization of multi-level logic networks which can solve these difficult circuit instances. The synthesis problem is transformed on the search problem. A search algorithm called Cartesian genetic programming (CGP) is applied to synthesize various difficult circuits. Conventional circuit synthesis usually fails for these difficult circuits; specific synthesis processes must be employed to obtain satisfactory results. We have found that CGP is able to implicitly discover new efficient circuit structures. Thus, it is able to optimize circuits universally, regardless their structure. The circuit optimization by CGP has been found especially efficient when applied to circuits already optimized by a conventional synthesis. The total runtime is reduced, while the result quality is improved further more.**

Keywords-logic synthesis, multi-level network, genetic programming

I. INTRODUCTION

In present time, logic synthesis experiences a newborn interest of researchers. Most of the currently used gate-level synthesis algorithms and processes have been established in 1980's and they are being used in today's commercial tools. Various types of circuit decompositions [1, 2, 3, 4], resynthesis processes [4, 5] and technology mapping [6, 7, 8] have been proposed at that time. Non-commercial open-source synthesis tools MIS [9], SIS [10] and MVSIS [11], implementing these algorithms were offered by Berkeley Logic Synthesis and Verification Group. Recently, their research shifted towards a different internal network representation: the And-Inverter Graphs (AIGs) [12]. These are more scalable than standard tabular (PLA) circuit nodes representations and new, more flexible synthesis and mapping algorithms may be applied upon these structures [13, 14, 15]. A synthesis tool ABC [16] implementing these algorithms came as a successor of SIS and MVSIS and its development still continues.

Basically, any present logic synthesis process is based on some kind of decomposition and resynthesis, followed by technology mapping. It has been shown that the decomposition shall satisfy the property of NPN-completeness [17] to reach satisfactory results, in terms of circuit area. A capability of XOR decomposition has been found essential for numerous circuits [18]. Unfortunately, the XOR decomposition is not implemented in ABC. We have found that a non-commercial tool BDS [19] performs all kinds of decompositions (AND, OR, XOR, MUX) efficiently.

Despite of all the late and recent developments in logic synthesis, current tools are not able to cope with newly emerging designs. Not only their ever-increasing size becomes a problem; there have been discovered very small circuits, for which synthesis tools produce extremely bad results [20]. Here the area of the synthesized circuits is of orders of magnitude higher than the optimum. Even though circuits presented in [20] were artificially constructed, their design is rather realistic, e.g., judging by their Rent's exponent. These circuits were originally constructed to test capabilities of look-up table (LUT) mappers. Lately we have found a huge class of real circuits for which synthesis crucially fails as well [21]. This failure is a problem of both academic (SIS, ABC) and commercial tools. If a large design is broken down to multiple smaller circuits and failures of this kind occur, we obtain an unacceptably large circuit without having any clue of the reason.

Summarized, up to our knowledge no available conventional synthesis process is able to efficiently discover disclosed structures and to create new, non-standard structures. The synthesis mostly fully relies on local optimizations [12]. Therefore, using some kind of global optimization may overcome drawbacks of present local synthesis algorithms. Genetic programming (GP) thus becomes an apparent option. The major feature of GP is that it does not employ any deterministic synthesis algorithm; all the optimizations are being done implicitly, without any structural biases.

In the area of evolutionary design various search-based algorithms have been proposed for circuit synthesis in the recent years. The main motivation was to discover novel circuit implementations that are unreachable by conventional synthesis methods. Cartesian Genetic Programming (CGP) is a method capable of efficient digital circuit synthesis purely on the basis of behavioral specifications [22, 23]. For example, CGP has discovered the implementations of small multipliers that require fewer gates than the best known conventional implementations (the mean gate number reduction is 18% [24]). However, a common drawback is that CGP does not scale well because of a very time consuming evaluation

978-1-4244-6610-8/10/$26.00 ©2010 IEEE

of candidate circuits. Although there are many papers on synthesis of ordinary circuits using CGP [24, 32, 33], the use of CGP for synthesis of conventionally hard to synthesize circuits was completely overlooked.

We present a method of circuit optimization/resynthesis using Cartesian genetic programming especially targeted to circuits that are difficult for conventional synthesis. Since the CGP is rather time-consuming, its most efficient application is as a post-synthesis. First, the circuit is processed by a standard synthesis, e.g., by ABC or BDS, and then it is further optimized by CGP. However, CGP reaches very good results even when starting from a non-optimized network. We show that CGP is able to implicitly perform many synthesis processes, like XOR decomposition. CGP is able to efficiently optimize circuits even having an artificially introduced "misleading" structure. The good structure is mostly discovered by CGP. As a result, CGP always significantly improves the processed network. We haven't found any conventional synthesis process reaching that quality results.

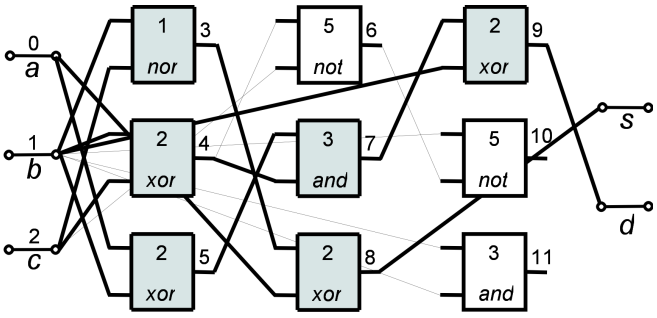
The paper is organized as follows: principles of the Cartesian Genetic Algorithm are presented in Section II., classes of hard-to-synthesize circuits are described in Section III. Details of the circuit optimization by CGP are given in Section IV. and experimental results in Section V. Section VI. concludes the paper.

II. Cartesian Genetic Programming

Cartesian Genetic Programming, introduced by Miller and Thompson [22], is a widely-used method for evolution of digital circuits and programs. In CGP, a candidate circuit is modeled as an array of u (columns) and v (rows) of programmable elements (gates). The number of circuit inputs, n_i, and circuit outputs, n_o, is fixed. Each node input can be connected either to the output of a node placed in the previous L columns or to some of the circuit primary inputs. Feedback is not allowed in the basic version of CGP. Each node is programmed to perform one of n_a-input functions defined in the set Γ (n_f denotes $|\Gamma|$). As Figure 1 shows, while the size of the chromosome is always fixed, the size of phenotype is variable (i.e. some nodes are not used). Each node is encoded using three integers (x, y, z) where x denotes the index for the first input, y denotes the index for the second input and z is the function code. Every individual is encoded using $u.v(n_a + 1) + n_o$ integers.

CGP operates with the population of 1+λ individuals (typically, λ = 4-20). The initial population is either randomly generated or seeded using a conventional design. Every new population consists of the best individual of the previous population and its λ offsprings created by a point mutation (h genes (integers) of the chromosome are randomly modified). In the case when two or more individuals have received the same fitness score in the previous generation, the individual which has not served as the parent will be selected as the new parent. This strategy is used to ensure the diversity of the population and is considered as a key attribute of CGP.

One of possible approaches to evaluation of a candidate individual (circuit) is to apply all possible assignments to the inputs and measure the number of resulting bits that are correct with respect to the specification. The higher number of correct resulting bits, the higher fitness score. Other approaches to fitness calculation utilize training/test sets or a kind of formal verification [25].



1,2,1; 1,2,2; 0,1,2; 4,2,5; 5,4,3; 3,0,2; 7,1,2; 1,6,5; 1,1,3; 8,9

Figure 1. Example of circuit encoding in CGP: $u = 3$, $v = 3$, $L = 3$, $n_i = 3$, $n_o = 2$, $n_a = 2$, Γ = {or (0), nor (1), xor (2), and (3), nand(4), not(5)}

III. Hard to Synthesize Circuits

A. LEKO and LEKU Benchmarks

Up to the knowledge of the authors, circuits "difficult" for conventional and current synthesis processes were introduced in [20] for the first time, originally to test the performance of LUT (look-up table) mappers. Even commercial tools were not able to manage the "misleading" benchmarks structure; the results were more than 500-times the optimum size. These circuits were called LEKO (Logic Examples with Known Optimum) and LEKU (Logic Examples with Known Upper Bound) benchmarks. The LEKO benchmarks are constructed by multiple replicating a relatively small circuit having n inputs and n outputs (n=5, 6), given as a Boolean network of two-input nodes. Optimum mapping into look-up tables with 4 inputs (4-LUTs) is known. After the circuit replication, there is a path from each input to each output. It has been proven that the optimum mapping of the entire circuit retains the mapping of the core circuit. Hence, the optimum multiplied circuit size is equal to the respective multiple of the original circuit size.

The LEKU benchmarks are constructed by collapsing the LEKO circuits into two-level sum-of-product (SOP) descriptions, followed by technology mapping. As a result of this process, the circuit's original structure is completely obscured. Consequently, the circuit description (network) size grows up significantly, since the obtained SOP is very large. Technology mapping run upon the SOP just decomposes the huge AND and OR gates, producing a network of numerous 2-input NAND gates.

Since the LEKU circuits are functionally equivalent to the LEKO ones, their expected size is known. However, even better designs could be theoretically obtained, if the circuits were synthesized and mapped properly. Therefore, the LEKO size is the upper bound imposed on the size of the LEKU circuits.

B. Realistic LEKU Benchmarks

The Cong & Minkovich's LEKU circuits are basically constructed by intentionally introducing a bad structure into the replicated core circuit; the circuit is made artificially large. This process may be performed on other, realistic benchmark circuits as well. Collapsing a multi-level network into a two-level circuit completely destroys the original structure, which is then very difficult to be recreated. Processing the circuit by a global BDD [29] does the same job. The size of the circuit usually significantly grows up, as in the LEKU case.

C. Difficult Standard Benchmark Circuits

Designers have objected to the Cong & Minkovich's LEKU circuits [20], since they are artificially constructed. However, we have discovered circuits from the standard LGSynth'93 benchmark set [27], which are difficult to synthesize as well. Particularly, the capability of XOR decomposition is required to synthesize these circuits properly. Without using XOR decomposition, the synthesized circuits are sometimes more than 25-times larger. Unfortunately, the XOR decomposition is not performed in all the available tools (SIS, ABC), except of BDS [19]. Many commercial tools are missing this ability as well.

D. Parity Benchmark Circuits

Recently we have encountered a new class of hard-to-synthesize realistic circuits. These circuits are constructed by appending a XOR tree to the circuit's outputs, to obtain one parity bit [28]. The upper bound of the area is the sum of the original circuit size and the size of the XOR tree. The circuit may be then resynthesized, with the hope of decreasing its size. We have found that conventional synthesis tools are not able to minimize the circuit size efficiently, unless it is collapsed into a two-level SOP network and resynthesized [28]. This process fully resembles the construction of the artificial LEKU benchmarks. The results of the resynthesis are spun between two extreme cases: at the "good" end, the circuit size is significantly reduced with respect to the upper bound, at the other end the size explodes [21]. The reason for the size explosion is the same as for the LEKU benchmarks – the obtained SOP is too large and the subsequent synthesis is not able to rediscover the original circuit structure. The need for XOR decomposition has been emphasized even more in these experiments. Tools not able to perform the XOR decomposition sometimes produced results 50-times larger than the upper bound.

E. Tautology and Near-Tautology Benchmarks

A different kind of artificially complex benchmarks can be created by generating large random SOPs. If the number of product terms in the SOP exceeds a particular threshold, the function turns into tautology. Notice that the SOP may be constructed of terms of higher dimensions, not only minterms. Functions described by SOPs with the number of terms near this tautology threshold are usually very simple – they are "near-tautologies". Two-level minimization [30] must be run in order to discover the true nature of functions described by these "big" SOPs. However, ABC and commercial tools do not do so. If this SOP (in form of a PLA or mapped into technology) is submitted to the synthesis, huge circuits are produced.

IV. Network Optimization by CGP

CGP is used according to its definition in Section II. In this paper we always use $v = 1$ and $L = u$. This setting imposes no structural restrictions on generated combinational circuits. CGP is seeded using a solution obtained from a conventional synthesis method. Experience shows that some redundancy is beneficial to the search process [25]. When CGP is applied on the result of conventional synthesis, then u equals the number of gates of the circuit resulting from conventional synthesis. Γ contains the gates conventional synthesis gate library. The particular CGP parameters are given in Section V.

Various approaches to fitness calculation can be utilized. For small circuits (with up to approx. 15 inputs), the fitness value of a candidate circuit is defined as:

$$\text{fitness} = B + (u.v - g)$$

where B is the number of correct output bits obtained as a response for all possible assignments to the inputs, g denotes the number of gates utilized in a particular candidate circuit and $u.v$ is the total number of gates available. The last term $(u.v - g)$ is considered only if the circuit behavior is perfect, i.e. $B = n_o.2^{ni}$; otherwise the fitness value equals B.

For larger circuits where the complete truth table evaluation is intractable, the candidate circuit is verified against the reference circuit using a SAT-based combinational equivalence checking algorithm. In order to evaluate the SAT-instances produced by the verification algorithm effectively, we have used a well-known SAT solver *Minisat* [26] which can be embedded into a custom application. The result of the verification algorithm is a Boolean value. If the value is negative (i.e., the candidate circuit and the reference circuit are not functionally equivalent) then the fitness score is zero. If the value is positive, the fitness value equals g.

In this paper we have used the second approach since only a few investigated circuits have less than 15 inputs.

V. Experimental Results

In this paper, optimization of combinational circuits given by a Boolean network of single-output internal nodes is considered. The nodes represent standard technology library gates. By network optimization we mean reducing the number of its nodes, thus reducing the area. The circuit delay is not considered, however it could be taken into account by modifying the CGP fitness function.

A. Experimental Setup

As for the synthesis process in ABC [16], we have used an extended *choice* resynthesis script, see Fig. 2, followed by technology mapping (ABC *map* command). All the circuits were mapped into the MCNC technology library [31] limited to 2-input gates only, for simplicity. The script was iterated 1000-times, to obtain better results.

```
fraig_store;
resyn; fraig_store;
resyn2; fraig_store;
resyn2rs; fraig_store;
share; fraig_store;
fraig_restore
```

Figure 2. Extended *choice* script

As for the CGP-based optimization, we have used the following parameters. Each CGP node can implement one of the eight basic functions Γ = {ID, NOT, AND, OR, XOR, NAND, NOR, XNOR}, where ID denotes the identity function. According to the preliminary experiments, we have chosen λ=2 and h=1 since this setup provided the best results.

We have used the following scheme for all the experiments: in the first step, the optimized circuit is converted from the BLIF file format to the CGP representation described in Section II. Since each circuit consists of the Γ gates only, the number of CGP columns corresponds to the number of circuit gates. Then, 50 independent experiments (runs) are performed for each circuit. A single run of CGP is terminated after 5 hours. Finally, the best result (i.e. the circuit with the minimal number of gates) is identified and converted to the BLIF file format. This file is then processed by the ABC command *sweep*, in order to remove possible identity gates and inverters.

Note that the CGP is usually terminated when the predefined limit of generations is reached. However, this criterion is unpractical in this case as the time of the fitness evaluation depends on the structural properties of the optimized circuit rather than the number of inputs and gates. The limit has been chosen as a trade-off between the required runtime and the obtained results. There are circuits that require only a fraction of the predefined runtime as well as there are circuits that can be further optimized. However, the same stopping criterion problem occurs in the ABC synthesis as well.

B. Processed Circuits

We present all the results in one summary table, so that the reader will be able to better relate the corresponding data. However, particular observations are discussed in separate subsections.

To test the CGP capabilities, we have processed a set of hard-to-synthesize benchmarks. Each circuit was generated by a specific synthesis process. All benchmarks belong to the categories described in Section III. The circuits were mapped into 2-input gates, as described in the previous Subsection.

Details are shown in Table I. After the benchmark name the numbers of its inputs (n_i), outputs (n_o) are shown. The description of the benchmark follows. The origin of the circuit is given, with the respective class presented in Section III (A-E). The benchmark name is retained from its benchmark set, the suffix "_p" indicates the parity tree attached to its output (Subsection III.C), the suffix "_c" means that the circuit was collapsed into a two-level description and mapped. The size of the benchmark, in terms of numbers of two-input gates is shown in the "*gates*" column.

C. CGP Used as a Primary Optimization Process

First, we have compared the CGP optimization with ABC, when run as primary optimization processes. The results are shown in Table I. columns "*ABC*" and "*CGP*". The ABC synthesis script used and the CGP parameters are described in Subsection V.A. The area obtained by CGP, in comparison with ABC is shown in the "*CGP vs. ABC*" column. It can be seen that CGP almost always outperforms ABC, sometimes significantly. The ABC resynthesis completely fails in these cases, while CGP is able to implicitly discover beneficial circuit structures. It is most apparent in the tautology and near-tautology examples. In cases where the difference between ABC and CGP is negligible, most probably the global optimum is approached by both methods.

There may be discussions regarding incomparable ABC and CGP runtimes. The ABC synthesis tool definitely requires less than 5 hours. However, repeating the ABC resynthesis script more than 1000 times usually does not bring any improvement; the synthesis quickly converges to a local minimum. So it was meaningless to repeat the resynthesis more times. The convergence curve for the *in6_p_c* benchmark is shown in Fig. 3. This circuit was selected because it has the slowest convergence of all the processed circuits.

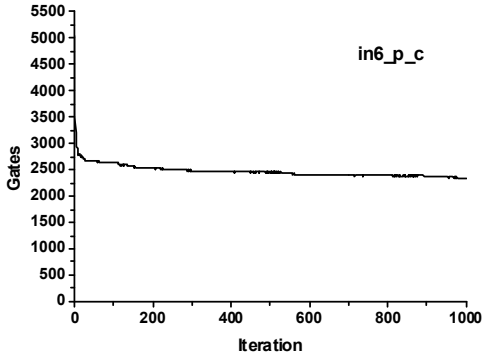


Figure 3. ABC convergence curve

On the other hand, CGP could yet produce better results, if it was run longer. The *in6_p_c* convergence curve, for a 5 hour run, is shown in Fig. 4.

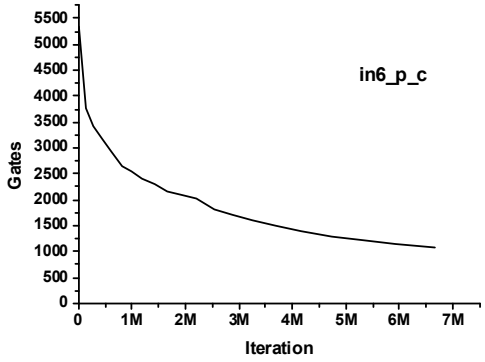


Figure 4. CGP convergence curve

D. CGP Used for a Post-Synthesis Optimization

Next, we have run the CGP to optimize circuits obtained by a conventional synthesis. Each of the presented benchmark categories requires a specific synthesis process, to obtain satisfactory results. Generally, it is the capability of XOR decomposition for C and D categories and collapsing for A, B, and E. We have processed the benchmarks by the respective processes and further minimized by the ABC script (V.A), to obtain the "best conventional solution". The numbers of gates are shown in Table I. "*Best conv.*" column.

Then, we have processed these circuits by CGP. The results are shown in the "*Best conv.+CGP*" column and the percentage improvement achieved is shown next. It can be seen that the conventional solution was almost always improved. The average improvement was 20%.

E. Proof of Deep Local Minima

We have tried to further optimize the CGP optimized circuits, by running the ABC resynthesis script. Surprising results were obtained, see columns "*CGP+ABC*" and "*Best conv.+CGP+ABC*". The results were almost always *deteriorated* by ABC. This gives us a hint that CGP is able to find a very deep local minimum in the circuit size, to refine the structure so that no other synthesis can improve it further more.

VI. Conclusions

We have presented a circuit optimization method based on Cartesian genetic programming (CGP). Experiments have shown that a significant area improvement can be reached this way. CGP is able to implicitly discover "good" structures in circuits, for which conventional synthesis completely fails. As a result, CGP can be efficiently used as a primary circuit optimization process, which, as we have found by processing numerous benchmark circuits, universally produces good results, regardless the original circuit structure.

The main drawback of the CGP optimization is a long runtime required to obtain satisfactory results, especially for large circuits. However, for the cost of runtime, CGP is able to produce results that conventional synthesis is never able to reach. The long runtime drawback may be partially compensated by running CGP as a post-synthesis process. The original circuit is first maximally reduced by a conventional synthesis and then optimized by CGP. As a consequence, the circuit size can be further reduced.

Acknowledgement

This research has been supported by MSMT under research program MSM6840770014 and MSM0021630528 and by the grant of the Czech Science Foundation GA102/09/1668 and GP103/10/1517.

References

[1] H. A. Curtis. A New Approach to the Design of Switching Circuits. Van Nostrand, Princeton, N.J., 1962.

[2] R. L. Ashenhurst. The decomposition of switching functions. In Proceedings of the International Symposium on the Theory of Switching, Part I 29, pages 74–116, 1957.

[3] R.K. Brayton and C. McMullen. The Decomposition and Factorization of Boolean Expressions. In Proceedings of the International Symposium on Circuits and Systems, pages 49–54, May 1982

[4] G. De Micheli. Synthesis and Optimization of Digital Circuits. McGraw-Hill, 1994

[5] S. Malik, E.M. Sentovich, R.K. Brayton, and A. Sangiovanni-Vincentelli. Retiming and resynthesis: Optimization of sequential networks with combinational techniques. IEEE Transactions on Computer-Aided Design, 10(1):74–84, January 1991

[6] C.W. Moon, B. Lin, H. Savoj, and R.K. Brayton. Technology Mapping for Sequential Logic Synthesis. In Proc. Int'l. Workshop on Logic Synthesis, North Carolina, May 1989.

[7] R. Murgai, Y. Nishizaki, N. Shenoy, R. K. Brayton, and A. Sangiovanni Vincentelli. Logic Synthesis for Programmable Gate Arrays. In Proceedings of the Design Automation Conference, pages 620–625, June 1990.

[8] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sangiovanni Vincentelli. Improved Logic Synthesis Algorithms for Table Look Up Architectures. In Proceedings of the International Conference on Computer-Aided Design, pages 564–567, November 1991.

[9] R. K. Brayton, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert R.Wang. MIS:AMultiple-Level Logic Optimization System. IEEE Transactions on Computer-Aided Design, CAD-6(6):1062–1081, November 1987.

[10] 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

[11] M. Gao, Jie-Hong Jiang, Y. Jiang, Y. Li, S. Sinha and R.K. Brayton: MVSIS, In the Notes of the International Workshop on Logic Synthesis, Tahoe City, June 2001

[12] A. Mishchenko, S. Chatterjee, R. K. Brayton, "DAG-aware AIG rewriting: a fresh look at combinational logic synthesis". In 43th Annual ACM IEEE Design Automation Conference, San Francisco, CA, USA, 2006, pp. 532-535.

[13] A. Mishchenko, S. Chatterjee, and R. Brayton, "Improvements to technology mapping for LUT-based FPGAs". IEEE TCAD, Vol. 26(2), Feb 2007, pp. 240-253.

[14] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts", Proc. ICCAD '07, pp. 354-361.

[15] A. Mishchenko, R. K. Brayton, and S. Chatterjee, "Boolean factoring and decomposition of logic networks", Proc. ICCAD'08, pp. 38-44.

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

[17] T. Sasao, Switching Theory for Logic Synthesis, Kluwer Academic Publishers, Norwell, MA, 1999

[18] P. Fišer and J. Schmidt, "The Observed Role of Structure in Logic Synthesis Examples", Proc. 18th of International Workshop on Logic and Synthesis 2009 (IWLS'09), Berkeley, California (USA), 31.7. - 2.8.2009, pp. 210-213

[19] C. Yang, M. Cieselski, V. Singhal, "BDS: A BDD-Based Logic Optimization System", in Proc. 37th DAC'00, 2000, p. 92.

[20] J. Cong and K. Minkovich, "Optimality study of logic synthesis for LUT-based FPGAs", IEEE Trans. CAD, vol. 26, pp. 230–239, Feb. 2007.

[21] P. Fišer and J. Schmidt, Small but Nasty Logic Synthesis Examples, Proc. 8th Int. Workshop on Boolean Problems (IWSBP'08), Freiberg, Germany, 18.-19.9.2008, pp. 183-190

[22] J. Miller, P. Thomson, "Cartesian Genetic Programming". In: Proc. of the 3rd European Conference on Genetic Programming EuroGP2000. LNCS 1802, Springer (2000), p. 121–132

[23] J. Miller, D. Job, V. Vassilev, "Principles in the Evolutionary Design of Digital Circuits - Part I". *Genetic Programming and Evolvable Machines.* 1(1): 8-35 (2000)

[24] V. Vassilev, D. Job, J. Miller, "Towards the Automatic Design of More Efficient Digital Circuits". In Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society, 2000, p. 151-160

[25] L. Sekanina, "Evolvable Components: From Theory to Hardware Implementations". Springer Verlag, Berlin 2004

[26] N. Een, N. Sorensson, „MiniSAT". http://minisat.se

[27] K. McElvain, "LGSynth93 Benchmark Set Version 4.0", May 1993.

[28] P. Fišer, P. Kubalík, H. Kubátová, "An Efficient Multiple-Parity Generator Design for On-Line Testing on FPGA", Proc. 11th Euromicro Conference on Digital Systems Design (DSD'08), Parma (Italy), 3. - 5.9.2008, pp. 94-99

[29] V. Bertacco and M. Damiani, "Disjunctive decomposition of logic functions", in *Proc. ICCAD'97*, 1997, pp. 78-82.

[30] R.K.Brayton, et al, Logic Minimization Algorithm for VLSI Synthesis, Norwell, MA: Kluwer Academic, 1984

[31] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide", MCNC, Technical Report 1991-IWLS-UGSaeyang, January, 1991.

[32] E. Stomeo, T. Kalganova, and C. Lambert, "Generalized disjunction decomposition for evolvable hardware," IEEE Transaction Systems, Man and Cybernetics, Part B, vol. 36, no. 5, pp. 1024–1043, 2006

[33] A. P. Shanthi and R. Parthasarathi, "Practical and scalable evolution of digital circuits," Applied Soft Computing, vol. 9, no. 2, pp. 618–624, 2009

TABLE I. RESULTS OF PROCESSED HARD BENCHMARKS

Name	n_i	n_o	*Origin and category*	*gates*	*ABC*	*CGP*	*CGP vs. ABC*	*Best conv.*	*Best conv. +CGP*	*Impr.*	*CGP +ABC*	*Best conv. +CGP +ABC*
9sym	9	1	[27], C	329	280	27	10%	57	48	16%	37	50
9sym_p_c	9	1	[27], D	217	214	37	17%	57	46	19%	38	49
alu1_p_c	12	1	[27], D	1085	795	52	7%	38	38	0%	57	38
b4_p_c	33	1	[27], D	9645	5008	5003	100%	279	98	65%	4535	104
big_pla	25	1	near-taut., E	15744	14940	24	0%	29	24	17%	24	24
c8_p_c	28	1	[27], D	1605	486	71	15%	53	51	4%	68	53
cc_p_c	21	1	[27], D	799	347	36	10%	54	36	33%	40	38
count_p_c	35	1	[27], D	1608	921	78	8%	57	54	5%	82	58
ex7_p_c	16	1	[27], D	1985	1392	719	52%	118	74	37%	726	87
i1_p_c	25	1	[27], D	759	397	37	9%	37	35	5%	36	35
in6_p_c	33	1	[27], D	5046	2386	798	33%	118	106	10%	759	111
LEKU-CB	25	25	[20], A	759	216	175	81%	235	181	23%	196	189
LEKU-CB_c	25	25	[20], coll.	699	214	178	83%	211	176	17%	182	189
LEKU-CD_c	25	25	[20], coll.	932	224	186	83%	195	177	9%	195	180
misex3c_p_c	14	1	[27], D	5869	4445	4444	100%	492	358	27%	4463	355
rd84	8	4	[27], C	713	395	31	8%	85	32	62%	35	33
s1238_p_c	32	1	[27], D	66633	52590	35116	67%	1916	897	53%	26726	935
s298_p_c	17	1	[27], D	2294	1483	52	4%	51	36	29%	61	41
s344_p_c	24	1	[27], D	3387	1910	76	4%	76	61	20%	80	59
s349_p_c	24	1	[27], D	3619	1950	79	4%	82	63	23%	80	73
s420.1_p_c	34	1	[27], D	4098	2521	2541	101%	80	80	0%	2281	81
s420_p_c	35	1	[27], D	2535	1175	141	12%	123	108	12%	148	109
signet_p_c	39	1	[27], D	49167	36974	45143	122%	8304	7453	10%	34991	7318
t481	16	1	[27], C	1263	420	21	5%	11	11	0%	15	11
taut1	25	1	taut., E	15397	14583	1	0%	1	1	0%	1	1
term1_p_c	34	1	[27], D	2397	918	80	9%	136	95	30%	75	101
ttt2_p_c	24	1	[27], D	13800	9828	13140	134%	66	55	17%	10414	62

Name – the benchmark name
n_i – number of benchmark inputs
n_o – number of benchmark outputs
Origin and category – see Section III
gates – number of gates in the original circuit
ABC – gates after ABC resynthesis
CGP – gates after CGP synthesis
CGP vs. ABC – percentage area comparison of GCP and ABC
Best conv. – the best solution found by conventional synthesis (gates)
Best conv. + CGP – "Best conv." post-synthesis by CGP (gates)
Impr. – percentual CGP improvement reached in "Best conv. + CGP"
CGP + ABC – post-synthesis of "CGP" results by ABC (gates)
Best conv. + CGP + ABC – post-synthesis of "Best conv. + CGP" results by ABC (gates)