Gate-Level Optimization of Polymorphic Circuits Using Cartesian Genetic Programming

Zbysek Gajda and Lukas Sekanina

Abstract— **Polymorphic digital circuits contain ordinary and polymorphic gates. In the past, Cartesian Genetic Programming (CGP) has been applied to synthesize polymorphic circuits at the gate level. However, this approach is not scalable. Experimental results presented in this paper indicate that larger and more efficient polymorphic circuits can be designed by a combination of conventional design methods (such as BDD, Espresso or ABC System) and evolutionary optimization (conducted by CGP). Proposed methods are evaluated on two benchmark circuits – Multiplier/Sorter and Parity/Majority circuits of variable input size.**

I. Introduction

Polymorphic digital circuits contain ordinary and polymorphic gates. Polymorphic gates are unconventional digital circuits which are able to change the logic function according to an external environment status (i.e., temperature, light, power supply voltage (V_{dd}) etc.) [1], [2], [3], [4], [5]. For example, the polymorphic AND/OR gate performs the AND function for 27 °C (in the first mode) or the gate performs the OR function for 125 °C (in the second mode). Figure 1 shows an example of a polymorphic circuit. Behavior of various polymorphic gates was demonstrated using simulations. The first example of fabricated polymorphic gate – the NAND/NOR gate controlled by V_{dd} – was presented by Stoica's group [3]. The six-transistor NAND/NOR gate operates as NOR for $V_{dd} = 1.8V$ and NAND for $V_{dd} = 3.3V$. The control of logic function via V_{dd} is unconventional but interesting for some applications [2], [5], [6]. The HP 0.5 micron CMOS technology was used for fabrication of the gate. Another NAND/NOR gate controlled by V_{dd} was developed and characterized by FIT (Faculty of Information Technology) Evolvable Hardware Group [5]. This eight-transistor gate operates as NAND for $V_{dd} = 5V$ and NOR for $V_{dd} = 3.3V$. The gate was fabricated using AMIS CMOS 0.7 micron technology. An experimental polymorphic reconfigurable ASIC was developed which contains configurable ordinary gates and polymorphic NAND/NOR gates controlled by V_{dd} [7]. This chip enables to investigate the electrical properties of polymorphic circuits and demonstrate the applications of polymorphic electronics.

Having polymorphic gates, researchers have begun to develop methods for synthesis of polymorphic circuits [8], [9], [10]. They have also integrated polymorphic gates into ordinary circuits to enhance their functionality [11], [12], [13], [6], [5]. Cartesian Genetic Programming (CGP) was used to evolve small polymorphic digital circuits [8], [10]. However, due to the scalability problem of the gate-level evolution, the most complex circuit evolved so far is the 4×3-bit Multiplier/7-bit Sorter [10]. On the other hand, the advantage of the gate-level evolutionary design is that it can provide very compact solutions (i.e. gate-optimized circuits).

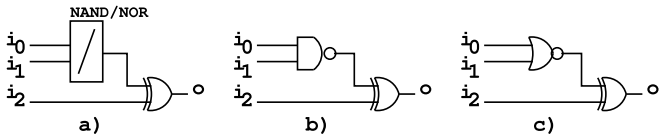

Fig. 1. Example of a polymorphic circuit: a) Scheme of a polymorphic circuit; b) Scheme of the circuit in the mode 1; c) Scheme of the circuit in the mode 2

In this paper, conventional methods developed for circuit synthesis (such as binary decision diagrams BDD [14], Espresso [15], and ABC[1] [16]) are combined with evolutionary algorithms in order to design and optimize larger polymorphic digital circuits. The general idea is to develop a circuit using conventional synthesis methods and then apply CGP to optimize the number of gates (as introduced for circuit evolution in [17]). Proposed methods are evaluated on two benchmark circuits: Multiplier/Sorter and Parity/Majority. In order to fairly compare the results, solutions will be sought in the form of circuits composed of two-input gates (inverters included). In addition to ordinary gates, we restrict ourselves to use only the NAND/NOR polymorphic gate controlled by V_{dd} because only this gate is available for a physical implementation. However, proposed methods can utilize an arbitrary set of polymorphic gates.

The rest of the paper is organized as follows. Section II formally describes the polymorphic circuit synthesis problem whose solution is the objective of this paper. Section III surveys the limits of CGP for design of polymorphic circuits. It also shows that slightly larger polymorphic circuits can be evolved using incremental evolution. In Section IV-A, BDDs are proposed to represent polymorphic circuits. Section IV-B describes the use of conventional methods Espresso and ABC (equipped with polymorphic multiplexers) to synthesize non-optimized polymorphic circuits. Section V deals with the optimization of polymorphic circuits which were obtained by using conventional methods. The optimization is performed

Zbysek Gajda and Lukas Sekanina are with the Department of Computer Systems, Faculty of Information Technology, Brno University of Technology, Czech Republic (email: {gajda, sekanina}@fit.vutbr.cz).

This work was partially supported by the Grant Agency of the Czech Republic under contract No. 102/06/0599 *Methods of polymorphic digital circuit design* and the Research Plan No. MSM 0021630528 – *Security-Oriented Research in Information Technology*.

[1] ABC is a System for Sequential Synthesis and Verification. For purpose of this paper, ABC is understood as a design method of sequential circuits.

using CGP, i.e. CGP is "seeded" with conventional designs. Discussion of obtained results is presented in Section VI. Conclusions are given in Section VII.

II. Polymorphic Circuit Synthesis Problem

A. Problem Formulation

Let $\Gamma^{(1)}$ denote a set of ordinary gates. Let $\Gamma^{(2)}$ denote a set of polymorphic gates. A polymorphic gate implements two[2] functions according to a control signal which can hold two different values. The gate is in *mode j* (and so performing the j-th function) in the case when j-th value of the control signal is activated. For purpose of this paper, we denote a polymorphic gate as X_1/X_2, where X_i is its i-th logic function. For example, NAND/NOR denotes the gate operating as NAND in the *mode 1* and as NOR in the *mode 2*. Note that ordinary gates can perform only one function; however, their functionality must be fully defined for each mode. For example, the conventional NAND gate considered for polymorphic circuits must perform the NAND function in the both modes (denoted as NAND/NAND). Let Γ denote a set of all gates, $\Gamma = \Gamma^{(1)} \cup \Gamma^{(2)}$.

A polymorphic circuit can formally be represented by a graph $G = (V, E, \varphi)$, where V is a set of vertices, E is a set of edges between the vertices, $E = \{(a, b)|a, b \in V\}$, and φ is a mapping assigning a function (gate) to each vertex, $\varphi : V \rightarrow \Gamma$. As usually, V models the gates and E models the connections of the gates. A circuit (and also its graph) is in the *mode j* in the case when all gates are in the *mode j*.

Given Γ and logic functions f_1 and f_2 required in different modes, the problem of the multifunctional circuit synthesis at the gate level is formulated as follows: Find a graph G representing the digital circuit which performs logic function f_1 in the first mode and logic function f_2 in the second mode. Additional requirements can be specified, e.g. to minimize the delay, the area, the power consumption etc. Unfortunately, this problem can not be approached by conventional synthesis methods directly since they do not allow representing polymorphic logic functions and manipulating with them.

B. Initial Solution

Figure 2 shows a straightforward approach to the implementation of a polymorphic circuit which works in $k = 2$ modes: The best known implementation for each mode is taken and the outputs are multiplexed by polymorphic multiplexers (or by standard multiplexers controlled by a sensor). Better results would be obtained by using such implementations which can share as much resources as possible (see the intersection in Figure 2).

A gate-level implementation of polymorphic multiplexer $pmux$ is shown in Figure 3. This implementation is based on the NAND/NOR gate. Its cost is c_{pmux} = 5 gates. We will use this implementation for the comparisons which will be performed in this paper. However, it is expected that a more compact and efficient transistor-level solution of $pmux$ will be available in the future.

[2]This can be naturally extended for k different functions.

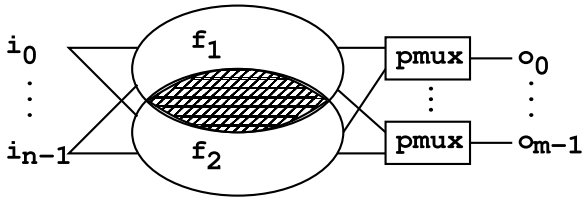

Fig. 2. Multiplexing circuits f_1 and f_2 by polymorphic multiplexers

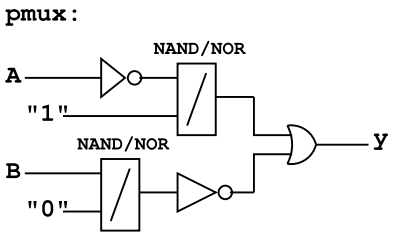

Fig. 3. Polymorphic multiplexer at the gate-level

III. Evolutionary Design of Polymorphic Circuits

A. Direct Evolution Using CGP

Cartesian Genetic Programming (CGP) introduced by Miller and Thompson [18], [19], [20] is a widely-used method for extrinsic evolution of digital circuits. CGP can easily be extended for gate-level evolution of polymorphic circuits [8]. Candidate circuits are modeled in a matrix of u (columns) $\times$ v (rows) of programmable 2-input elements (gates). The number of inputs, n, and outputs, m, is fixed. Each gate input can be connected either to the output of a gate placed in the previous L columns or to some of circuit inputs. The L-back parameter, in fact, defines the level of connectivity and thus reduces/extends the search space. For example, if L=1 only neighboring columns may be connected; if $L = u$, the full connectivity is enabled. Feedback is not allowed. Each gate can be programmed to perform one of functions defined in the set Γ which will contain polymorphic functions in our task.

The fitness function for synthesis of polymorphic circuits extends the fitness function utilized to evolve digital circuits [21]. Let f_1 denote the multiplication function and f_2 denote the sorting function in the case that the goal is to evolve a Multiplier/Sorter circuit. The fitness value is defined as:

$$fitness = B_1 + B_2 + (u.v - z) \quad (1)$$

where B_1 (resp. B_2) is the number of correct output bits for f_1 (resp. f_2) obtained as response for all possible input combinations, z denotes the number of gates utilized in a particular candidate circuit and $u.v$ is the total number of programmable gates available. The last term is considered only if the circuit behavior is perfect in the both modes; otherwise $u.v - z = 0$.

Table I summarizes existing results for the Multiplier/Sorter problem obtained using CGP in our previous work [10]. For all problems, 10 runs were executed per experiment, the population size was 15 and up to 100 million generations were produced in each run. The 7-input Multiplier/Sorter is the most complex polymorphic module evolved so far. The limit of the method was probably reached in terms of generated circuit complexity.

TABLE I

PARAMETERS AND RESULTS OF CGP FOR THE MULTIPLIER/SORTER PROBLEM ACCORDING TO [10]. GATES IN Γ ARE NUMBERED AS: (1) NAND/NOR, (2) AND, (3) OR, (4) XOR, (5) NAND, (6) NOR, (7) NOT A, (8) NOT B, (9) MOV A AND (10) MOV B, WHERE MOV DENOTES THE IDENTITY OPERATION.

Multiplier/Sorter	**2×2/4b**	**3×2/5b**	**3×3/6b**	**4×3/7b**
$u \times v$	10×12	100×1	120×1	16×16
L-back	1	100	120	16
Mutation (genes)	1	2	4	4
Gate set	1, 2, 9, 10	1–4, 9, 10	1–10	1, 2, 9, 10
Successful runs	100%	100%	90%	30%
Generations (avg.)	52,580	854,900	26,972,648	62,617,151
Min. # of gates	23	30	52	113

Table II compares the number of gates required for CGP with polymorphic multiplexing (according to Section II-B). The best known Multipliers (according to [17]) and Sorters (according to [22]) that were designed separately have been chosen as modules for polymorphic multiplexing. No sharing of gates is assumed herein. It is evident that in some cases the evolved circuits are more gate-efficient than the circuits multiplexing independent implementations despite the fact that only the NAND/NOR gate is considered. Including other polymorphic gates could even lead to better results.

TABLE II

COMPARISON OF THE IMPLEMENTATION COST (# OF GATES) FOR MULTIPLIER/SORTER IMPLEMENTED USING (A) MULTIPLEXING INDEPENDENT SOLUTIONS AND (B) CGP

Inputs	**Multiplier**	**Sorter**	**Multiplexing**	**CGP**
2 + 2	7	10	$17 + 4c_{pmux} = 37$	23
3 + 2	13	18	$31 + 5c_{pmux} = 56$	30
3 + 3	23	24	$47 + 6c_{pmux} = 77$	52
3 + 4	38	32	$70 + 7c_{pmux} = 105$	113

Table III summarizes results for Majority/Parity benchmark. CGP parameters are identical with the experiments reported in Table I except the mutation rate (3 genes). In this case, CGP can evolve Majority/Parity benchmark circuits with up to 13 inputs.

B. Incremental evolution

Evolutionary design of larger gate-level circuits is usually performed using modular CGP [23] or incremental evolution [24], [25], [26]. Figure 4 shows one of the approaches to the incremental evolution. The n-input/m-output circuit can be decomposed to m modules. Each module implements an n-input/1-output function. By doing so, the problem becomes easier for the evolution. This incremental evolution scheme was applied to evolve the 4×4-bit Multiplier/8-bit Sorter. The circuit was decomposed to 8 modules and each of them was evolved separately using CGP (parameters: 10 runs, 15 individuals in the population, 100 million generations per run). From Table IV, it can be seen that the resulting 4×4-bit Multiplier/8-bit Sorter contains 289 gates (sum of module costs). Then, a single circuit was composed of the modules and optimized using CGP (parameters remain unchanged) with the fitness function according to eq. (1). The optimized solution consists of 245 gates.

Unfortunately, the approach does not scale for larger problem instances. No solution was obtained for circuit o_5 of the 5×4-bit Multiplier/9-bit Sorter. Hence, other algorithms have to be employed to obtain large (Multiplier/Sorter) polymorphic circuits.

TABLE III

PARAMETERS AND RESULTS OF CGP FOR MAJORITY/PARITY PROBLEM. THE GATE SET INCLUDES NAND/NOR, AND, OR, XOR, NAND, NOR, NOT, MOV, WHERE MOV DENOTES THE IDENTITY OPERATION.

Majority/Parity	**7b**	**9b**	**11b**	**13b**
$u \times v$	80×1	120×1	120×1	160×1
L-back	80	120	120	160
Successful runs	100%	90%	50%	10%
Generations (avg.)	766,362	4,762,745	8,145,890	9,712,501
Min. # of gates	25	42	61	80

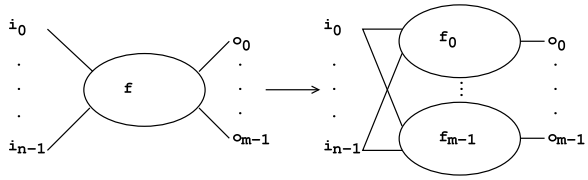

Fig. 4. Incremental evolution: decomposition of a function to m modules

TABLE IV

INCREMENTAL EVOLUTION OF 4×4 BIT MULTIPLIER/8-BIT SORTER.

Output of m./s.n. 4×4/8b	o_0	o_1	o_2	o_3
Elements	160	160	160	160
Generations (avg.)	3M	0.9M	0.9M	16.6M
Success	100%	100%	100%	100%
Min. gates	8	20	32	44
Output of m./s.n. 4×4/8b	o_4	o_5	o_6	o_7
Elements	160	100	160	100
Generations (avg.)	72.4M	46.9M	9.4M	1.5M
Success	60%	20%	100%	100%
Min. gates	56	71	42	16

IV. CONVENTIONAL SYNTHESIS

In order to overcome the scalability limits of evolutionary design and to maximize sharing of resources (compacting target circuits), three conventional approaches are tested in this section. The main issue is how to include polymorphic gates to conventional circuits.

A. Binary Decision Diagrams

The representation of a circuit using Binary Decision Diagram (BDD) [14] implies the implementation which is based on multiplexers whose selection signals are controlled by input variables. In the case of polymorphic circuits, terminal nodes of the decision diagram can be implemented using polymorphic functions. In comparison to polymorphic multiplexing (Section II-B), this method represents in principle a quite different approach to introducing polymorphic gates to conventional circuits.

The construction requires two steps. Firstly, a complete binary tree is constructed with $n-1$ levels, where n is the number of inputs. The nodes are *"if-then-else"* conditions which are decided by the input variables. These nodes represent 2-input multiplexers. Secondly, terminals are connected to the low-level nodes. The values of terminals are defined by a truth table. Each terminal value defines a single polymorphic circuit. For example (see Figure 5a), consider 3-bit Majority/Parity circuit. For the input vector 000, the circuit must return *"0"* in both modes. For the input vector 001, it must return *"0"* (majority function) or *"1"* (parity function). The input combinations 000 and 001 determine the terminal value according to a conversion matrix (see Figure 5b) and equation $s = 2^3.s_{21} + 2^2.s_{20} + 2^1.s_{11} + 2^0.s_{10}$. In this case, the value is 8 (see Figure 5c). The terminal defines a polymorphic circuit which outputs *"0"* for *mode 1* and *identity* for *mode 2*. The input vector 00 of the transformed truth table (see Figure 5c) defines a position of the terminal in the binary tree.

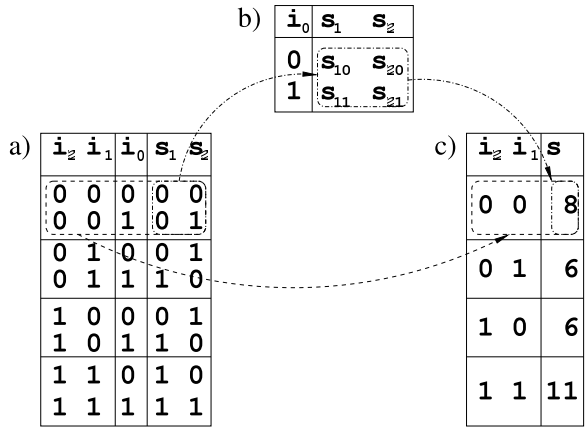

Fig. 5. Transformation process of 3-bit Majority/Parity truth table (s_1 denotes the majority; s_2 denotes the parity): a) Truth table before transformation; b) Transformation matrix for all quarters of the table; c) Transformed truth table

Figure 6 shows reduced BDD and its implementation using polymorphic gates. In order to reduce the size of BDD, we applied (classical) algorithms which allowed a reduction of identical terminals, sub-diagrams and redundant nodes. The nodes are implemented as multiplexers and the terminal nodes are implemented as independent polymorphic circuits.

Table V shows results of BDD-based polymorphic circuit synthesis for Multiplier/Sorter problem. By "Gates" we mean common 2-input gates, NAND/NORs, 2-input multiplexers and inverters.

TABLE V
BDD DESIGN RESULTS

Multiplier/Sorter	3×2/5b	3×3/6b	4×3/7b	4×4/8b
Nodes	37	79	135	253
Terminals	10	11	11	12
Gates	50	94	150	269

Note that this type of BDD can be understood as a MTBDD (Multi-Terminal Binary Decision Diagram) [14].

B. Espresso and ABC

Espresso [15] and ABC [16] are conventional circuit synthesis methods. We applied them according to Figure 2 with the aim of minimizing the number of gates in both modules, sharing as much gates as possible among the modules and minimizing the number of outputs that have to be equipped with polymorphic multiplexers. Table VI shows results of Espresso and ABC synthesis for Multiplier/Sorter benchmark. All circuits were implemented using two-input gates AND, OR, NAND, NOR, XOR, NAND/NOR and inverter (uniform cost considered).

TABLE VI
RESULTS OBTAINED USING ESPRESSO AND ABC

Multiplier/Sorter	3×2/5b	3×3/6b	4×3/7b	4×4/8b
Espresso: gates	168	419	960	2309
ABC: gates	61	119	198	359

V. EVOLUTIONARY OPTIMIZATION OF CONVENTIONAL DESIGNS

By comparing Tables I, V, and VI, it can be seen that evolutionary design provides more compact circuits than conventional methods. However, the results of conventional designs were not optimized (in terms of polymorphic gates utilization). Hence, the evolutionary approach was used to reduce the number of gates in the conventional solutions.

In order to optimize polymorphic circuits obtained by conventional synthesis methods, all these circuits are converted to the CGP representation (with topology $k \times 1$ where k is total number of elements) and used for *"seeding"* the CGP. A multiplexer is converted into four 2-input gates (in BDD design), 3-input AND-gate is converted into two 2-input gates (in Espresso designs) etc. CGP operates with the population size of 15 individuals. The mutation operator modifies 7 integers in the chromosome on average. The fitness function is defined using eq. 1. In order to make the evaluation as short as possible, a candidate circuit is immediately left unevaluated (i.e. $fitness = 0$) when it fails for an input test vector. For each benchmark problem, the CGP optimization was performed 10 times.

Table VII shows a comparison of results obtained by applying CGP on 4×4-bit Multiplier/8-bit Sorter circuits created by BDD, Espresso and ABC synthesis. We can observe

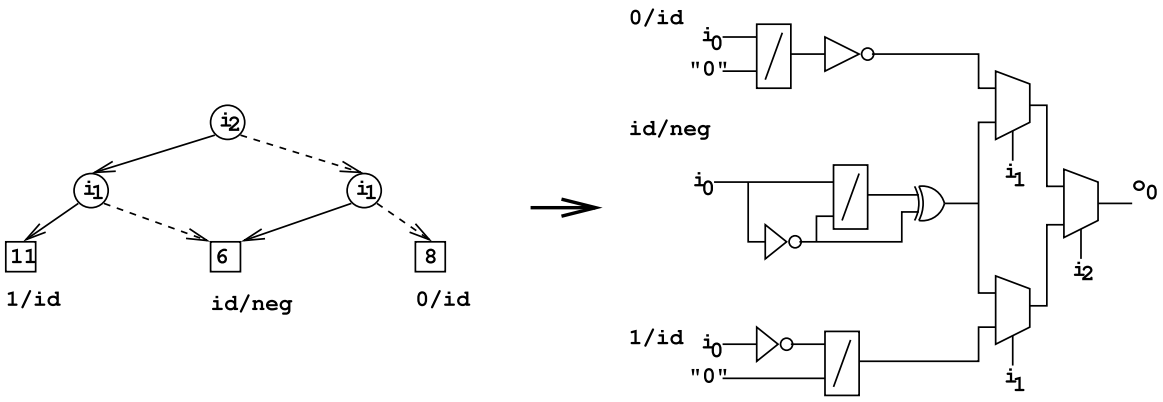

Fig. 6. BDD and a corresponding polymorphic circuit for the 3-bit Majority/Parity.

that the solutions are significantly different in the number of gates (see Min. gates). The best result was obtained by ABC synthesis. So, we used ABC as basis for further optimization of other instances of Multiplier/Sorter benchmark problem. It can be seen from Table VIII that ABC followed by CGP optimization can provide better results than other approaches (including direct CGP evolution); however, the results are better only for larger circuits when the number of inputs is 7 or higher. The ABC followed by CGP optimization was also utilized to develop Majority/Parity circuits (Table IX). By comparison of Table IX and Table III, we can see that the direct CGP design provides better results.

TABLE VII
RESULTS OF THE CGP OPTIMIZATION OF 4×4-BIT MULTIPLIER/8-BIT SORTER "SEEDED" BY CONVENTIONAL METHODS

Method	BDD	Espresso	ABC
Elements (k)	1043	2330	375
Initial gates	1028	2309	359
Generations	100M	100M	100M
Max. gates	407	697	232
Min. gates	**355**	**616**	**205**
Avg. gates	385.7	646.3	218.5
Avg. optimization	38%	28%	61%
Polymorphic gates	8%	3%	22%
Design runtime [s]	345	1	3
Optimization runtime [s]	154,015	361,394	21,176

TABLE VIII
RESULTS OF THE ABC FOLLOWED BY CGP OPTIMIZATION FOR VARIOUS INSTANCES OF THE MULTIPLIER/SORTER CIRCUIT

Multiplier/Sorter	3x2/5b	3x3/6b	4x3/7b	4x4/8b
Elements (k)	71	131	212	375
Initial gates	61	119	198	359
Generations	10M	10M	100M	100M
Max. gates	43	82	135	232
Min. gates	**36**	**71**	**110**	**205**
Avg. gates	38.8	76.3	121.4	218.5
Avg. optimization	64%	64%	61%	61%
Polymorphic gates	27%	22%	25%	22%

TABLE IX
RESULTS OF THE ABC FOLLOWED BY CGP OPTIMIZATION FOR VARIOUS INSTANCES OF THE MAJORITY/PARITY CIRCUIT

Majority/Parity	7b	9b	11b	13b
Elements (k)	41	60	81	114
Initial gates	39	58	79	112
Generations	1M	1M	10M	10M
Max. gates	33	53	72	95
Min. gates	**29**	**45**	**69**	**90**
Avg. gates	31	48.3	71.1	91.8
Avg. optimization	79%	83%	90%	82%
Polymorphic gates	18%	19%	18%	19%

VI. DISCUSSION

Direct polymorphic circuit evolution using CGP is not scalable. For Multiplier/Sorter circuits, the limit seems to be in 7 inputs. On the other hand, CGP can generate very compact solutions for small problem instances. Although the incremental evolution can generate larger circuits than CGP, it moves the scalability limit only partially (to the 8 inputs for Multiplier/Sorter benchmark problem).

Experimental results indicate that combining conventional methods with evolutionary optimization can lead to compact polymorphic circuits. When BDDs are used, polymorphic gates are connected towards the inputs of the circuit. Espresso or ABC are employed to find such implementations in which gates of f_1 and f_2 are reused as much as possible. Polymorphic gates situated close to outputs of the circuit ensure multiplexing the modules according to the external control. The most compact circuits were obtained for the 4×4-bit Multiplier/8-bit Sorter by BDD (269 gates); ABC requires 359 gates and Espresso 2309 gates. The highest number of gates produced by Espresso is mainly due the fact that many-input gates have to be transformed into 2-input gates.

Consequent optimization of these circuits, which was performed by *"pre-seeded"* CGP, shows that results of ABC can be improved by more than 35%, reaching thus 205 gates for the 4×4-bit Multiplier/8-bit Sorter (the best result) and 110 gates for the 4×3-bit Multiplier/7-bit Sorter (which is better result than direct evolutionary design using CGP).

Another important property of CGP optimization applied on circuits created by ABC is that the number of polymorphic gates utilized in resulting circuits is high - more than 20% in comparison to 8% for BDD and 3% for Espresso. For example, 16 NAND/NOR gates (in 8 polymorphic multiplexers) have to be used in the 4×4-bit Multiplier/8-bit Sorter in order to start the ABC synthesis. The resulting circuit (after CGP optimization) contains 44 NAND/NOR gates. It is supposed that the high proportional representation of polymorphic gates causes the compact implementation (only 205 gates).

More comprehensive validation of this concept on a larger set of benchmark circuits is needed. A potential problem is that CGP optimization is based on testing all possible input vectors which is not scalable.

VII. Conclusions

In this paper, a new method was proposed for design of polymorphic gate-level circuits. We surveyed existing approaches to the polymorphic circuit design and showed their limitations. Conventional methods (such as BDD, Espresso and ABC) were extended to support polymorphic gates. Experimental results presented in this paper indicate that combination of conventional methods and evolutionary optimization conducted by CGP can lead to larger and more efficient polymorphic circuits. Our future work will be focused on validating the proposed methods on more complex benchmark problems. We will also investigate the effect of various fitness functions and a multiobjective approach to the optimization.

References

[1] A. Stoica, R. S. Zebulum, and D. Keymeulen, "Polymorphic electronics," in *Proc. of Evolvable Systems: From Biology to Hardware Conference*, ser. LNCS, vol. 2210. Springer, 2001, pp. 291–302.

[2] A. Stoica, R. S. Zebulum, D. Keymeulen, and J. Lohn, "On polymorphic circuits and their design using evolutionary algorithms," in *Proc. of IASTED International Conference on Applied Informatics AI2002*, Insbruck, Austria, 2002.

[3] A. Stoica, R. Zebulum, X. Guo, D. Keymeulen, I. Ferguson, and V. Duong, "Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration," *IEE Proc.-Comp. Digit. Tech.*, vol. 151, no. 4, pp. 295–300, 2004.

[4] R. S. Zebulum and A. Stoica, "Four-Function Logic Gate Controlled by Analog Voltage," *NASA Tech Briefs*, vol. 30, no. 3, p. 8, 2006.

[5] R. Ruzicka, L. Sekanina, and R. Prokop, "Physical demonstration of polymorphic self-checking circuits," in *Proc. of 14th IEEE International On-Line Testing Symposium*. IEEE, 2008, pp. 31–36.

[6] L. Starecek, L. Sekanina, and Z. Kotasek, "Reduction of test vectors volume by means of gate-level reconfiguration," in *Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop*. IEEE Computer Society, 2008, pp. 255–258. [Online]. Available: http://www.fit.vutbr.cz/research/view_pub.php?id=8603

[7] L. Sekanina, R. Ruzicka, Z. Vasicek, R. Prokop, and L. Fujcik, "Repomo32 – new reconfigurable polymorphic integrated circuit for adaptive hardware," in *2009 IEEE Workshop on Evolvable and Adaptive Hardware*. IEEE Computational Intelligence Society, 2009.

[8] L. Sekanina, "Evolutionary design of gate-level polymorphic digital circuits," in *Applications of Evolutionary Computing*, ser. LNCS, vol. 3449. Lausanne, Switzerland: Springer Verlag, 2005, pp. 185–194.

[9] W. Luo, Z. Zhang, and X. Wang, "Designing polymorphic circuits with polymorphic gates: a general design approach," *IET Circuits, Devices & Systems*, vol. 1, no. 6, pp. 470–476, 2007.

[10] L. Sekanina, L. Starecek, Z. Kotasek, and Z. Gajda, "Polymorphic gates in design and test of digital circuits," *International Journal of Unconventional Computing*, vol. 4, no. 2, pp. 125–142, 2008.

[11] R. S. Zebulum and A. Stoica, "Multifunctional Logic Gates for Built-In Self-Testing," *NASA Tech Briefs*, vol. 30, no. 3, p. 10, 2006.

[12] ——, "Ripple Counters Controlled by Analog Voltage," *NASA Tech Briefs*, vol. 30, no. 3, p. 2, 2006.

[13] L. Sekanina, "Evolution of Polymorphic Self-Checking Circuits," in *Proc. of the 7th Conf. on Evolvable Systems: From Biology to Hardware*, ser. LNCS, no. 4684. Wuhan, China: Springer, 2007, pp. 186–197.

[14] R. Drechsler and B. Becker, *Binary Decision Diagrams: Theory and Implementation*. Kluwer Academic Publishers, Boston, USA, 1998.

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

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

[17] V. Vassilev, D. Job, and J. F. Miller, "Towards the automatic design of more efficient digital circuits," in *Proc. of the 2nd NASA/DoD Workshop of Evolvable Hardware*. Los Alamitos, CA, US: IEEE Computer Society, 2000, pp. 151–160.

[18] J. Miller and P. Thomson, "Cartesian Genetic Programming," in *Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*, ser. LNCS, vol. 1802. Springer, 2000, pp. 121–132.

[19] J. Miller, D. Job, and V. Vassilev, "Principles in the Evolutionary Design of Digital Circuits – Part I," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1, pp. 8–35, 2000.

[20] J. A. Walker, J. F. Miller, and R. Cavill, "A multi-chromosome approach to standard and embedded cartesian genetic programming," in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, vol. 1. ACM Press, 2006, pp. 903–910.

[21] T. Kalganova and J. Miller, "Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness," in *The First NASA/DoD Workshop on Evolvable Hardware*. Pasadena, California: IEEE Computer Society, 19-21 1999, pp. 54–63.

[22] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.

[23] J. A. Walker and J. Miller, "The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 4, pp. 397–417, 2008.

[24] J. Torresen, "A scalable approach to evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 3, no. 3, pp. 259–282, 2002.

[25] 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.

[26] T. Kalganova, "Bidirectional incremental evolution in extrinsic evolvable hardware," in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Silicon Valley, USA, July 2000, pp. 65–74.