Two-Step Evolution of Polymorphic Circuits for Image Multi-Filtering

Lukas Sekanina, Vojtech Salajka and Zdenek Vasicek
Brno University of Technology, Faculty of Information Technology, IT4Innovations Centre of Excellence
Brno, Czech Republic
Email: sekanina@fit.vutbr.cz

***Abstract*—This paper proposes to implement multifunctional image filters using multifunctional gates such as polymorphic gates or multiplexed ordinary gates. The design procedure is based on evolutionary design and optimization conducted using Cartesian genetic programming (CGP). Because of the complexity of the problem the design is decomposed to two phases. In the first step, a multifunctional filter is evolved at the register-transfer level (RTL) using a set of processing elements containing functions such as minimum/maximum, minimum/average etc. over two pixels. In the second step, gate-level implementations of the processing elements utilized in evolved filters are designed and optimized using CGP in combination with conventional logic synthesis tools. It is shown that resulting filters exhibit good filtering capabilities. They are also area-efficient in comparison with solutions based on multiplexing of ordinary filters.**

I. Introduction

Obtaining flexibility, adaptation and multifunctionality directly at the hardware level is one of the most important goals we can now observe in the reconfigurable hardware field [1]. These objectives are usually achieved by hardware reconfiguration which is performed by supplying a new configuration bit stream to a configuration memory. The configuration bit stream then activates relevant configuration signals and switches which, when activated, establish new circuit connections and thus new functionality directly in hardware. The main advantage is that many new configurations (even those unanticipated during the design time) can be created in the runtime. However, the associate area/latency overhead caused by the reconfiguration infrastructure (configuration switches, network and memory) makes this type of systems very expensive in applications, where only a few preselected configurations have to be employed.

One of possible approaches to achieving a low cost reconfiguration could be based on multifunctional gates. They have special physical structures enabling to behave differently with respect to external conditions. Recently developed graphene logic gates based on graphene pn junctions are capable of performing several logic functions just by adjusting some control voltages [2]. Various logic functions can also be provided by polymorphic CMOS gates introduced ten years ago. Their control is implemented via control voltages, power supply voltage (V_{dd}) or temperature [3], [4]. As the V_{dd}-based control does not require any additional wires the use of polymorphic gates could reduce interconnecting networks in reconfigurable chips significantly.

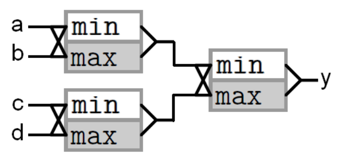

Fig. 1. Multifunctional circuit composed of three minimum/maximum elements calculating $y = min(a,b,c,d)$ in the first mode and $y = max(a,b,c,d)$ in the second mode.

In this paper, it is proposed to combine low-level image processing functions such as noise elimination and edge detection into one compact multifunctional circuit - multifunctional filter. The resulting circuit is then composed of multifunctional gates connected using a network which is fixed and predesigned. Predefined filtering functions are activated using a suitable setting of control signals. It is supposed that a significant reduction in the area and routing can be obtained in comparison to a conventional implementation which is typically based on multiplexing of conventional filtering circuits.

Designing a compact multifunctional filter, which contains multifunctional gates is a challenging task. In this paper, we will consider multifunctional circuits performing just two different functions (F_1 and F_2) and thus operating in two different modes (although extending the concept for more modes is straightforward). Formally, a single network containing multifunctional (and ordinary gates if needed) has to be constructed such that F_1 is implemented in the first mode and F_2 is implemented in the second mode (Fig. 1). Polymorphic NAND/NOR gate is a typical elementary component of such circuits [3], [4].

Various synthesis and optimization methods for multifunctional gate-level circuits, including evolutionary design and optimization, were proposed [5], [6]. Evolutionary design has also been applied to design image filters suppressing a given type of noise [7], [8]. However, image filter evolution has been conducted at the functional level since the gate-level representation is too low-level to evolve reasonably working filters.

In order to evolve multifunctional image filters, we propose to apply two-step evolution [9]. In the first step, multifunctional filter is evolved at the register-transfer level (RTL) using a set of processing elements containing functions such

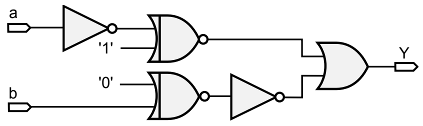

Fig. 2. A polymorphic multiplexer using NAND/NOR gates

as minimum/maximum, minimum/average etc. over two pixels [10]. It is assumed that such functions can easily be composed using multifunctional gates. In the second step, gate-level implementations of multifunctional processing elements utilized in evolved filters are designed and optimized using evolution. Both design tasks will be carried out using Cartesian genetic programming (CGP). We will demonstrate on three case studies that resulting filters are functionally comparable with conventionally designed filters. The main contribution of this paper is that it shows that evolved filters are area-efficient in comparison with solutions based on multiplexing of ordinary filters.

II. Multifunctional Circuits

Logic function of multifunctional gates can be selected by various means, including external logic signals (such as the selector in multiplexers) or external voltage signals. When the control variable is V_{dd} or temperature then there is no additional wire needed. For instance, a 6-transistor NAND/NOR gate controlled by V_{dd} was fabricated in a 0.5-micron HP technology [4]. Another NAND/NOR gate controlled by V_{dd} was utilized in the REPOMO32 chip [11]. A graphene gate which is configured by external voltages is capable of performing 8 different functions in its basic mode [2].

Theoretical works on polymorphic networks such as the completeness theory can be found in [12], [13]. Paper [5] surveys the methods proposed to design multifunctional circuits.

Polymorphic multiplexing is the most straightforward design approach. It employs a polymorphic multiplexer – a component which propagates signal A in the first mode of multifunctional gates and signal B in the second mode of multifunctional gates 2 (Fig. 2). Consider that a target polymorphic circuit has to implement F_1 and F_2. A conventional approach is used to synthesize a circuit implementing F_1 and another circuit implementing F_2 independently. The outputs of the circuits are then multiplexed using polymorphic multiplexers. In order to reduce the number of gates, the goal of synthesis can be to maximize the number of gates that are shared by both circuits and minimize the number of outputs that have to be equipped with polymorphic multiplexers.

CGP is capable of evolving very area-efficient multifunctional circuits, however, the approach evaluating 2^n input assignments for n-input circuits is not scalable [14], [5], [15]. Only small polymorphic FIR filters were designed using this method [16]. In order to overcome this very time consuming evaluation, we will apply a functional equivalence checking algorithm in Section VI as suggested in [6] .

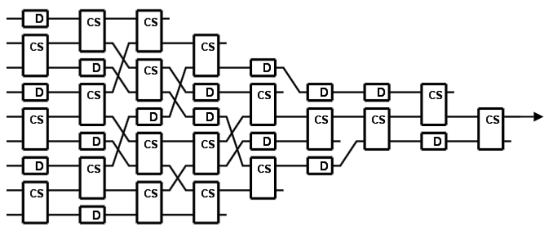

Fig. 3. The 9-input median circuit

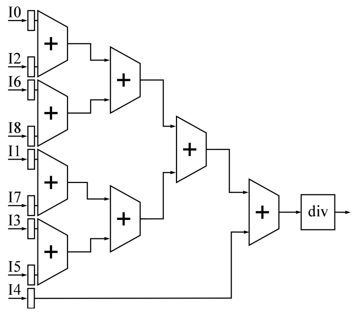

Fig. 4. The 9-input averaging circuit

III. Conventional and Multifunctional Image Filters

The target application for this work is a multifunctional image filter operating with a 3×3-pixel kernel. Every image filter is considered as a digital circuit of nine 8-bit inputs (the 3×3-pixel kernel) and a single 8-bit output, which processes grayscale (8-bits/pixel) images. CGP is employed to devise a filter composed of ordinary and multifunctional components capable of suppressing a given type of noise in the first mode and another type of noise in the second mode. In addition to noise filtering (in particular, shot noise and Gaussian noise elimination), edge detection, dilatation and erosion are other target functions.

There are well-established conventional methods allowing us to suppress a given type of noise. The shot noise (also called the salt-and-pepper noise) is usually suppressed by a (nonlinear) median filter which calculates the median value from the nine input pixels. Fig. 3 shows the area-optimal (pipeline) implementation which consists of compare-and-swap (CS) components and registers (D) [17]. The CS component calculates the minimum and maximum out of the two input values. The Gaussian noise elimination is based on a linear convolution – a simple averaging filter is shown in Fig. 4. In case of edge detection, Sobel detector will be used in our case study. An ideal dilatation filter calculates the maximum out of the input pixels and similarly, an ideal erosion filter calculates the minimum out of the input pixels [18].

A straightforward implementation of a multifunctional filter will multiplex an ordinary circuit created to eliminate the first type of noise and another ordinary circuit created to eliminate the second type of noise. As structures of both filters are

TABLE I
LIST OF FUNCTIONS FOR EVOLUTION OF IMAGE FILTERS AND THEIR IMPLEMENTATION COST.

code	function	description	gates	area
`const`	c	constant	0	0.0
`ident`	x	identity	0	0.0
`or`	$x \vee y$	bitwise OR	8	10.7
`nor`	$\neg(x \vee y)$	bitwise OR inverted	8	8.0
`and`	$x \wedge y$	bitwise AND	8	10.7
`nand`	$\neg(x \wedge y)$	bitwise AND inverted	8	8.0
`xor`	$x \oplus y$	bitwise XOR	8	16.0
`nxor`	$\neg(x \oplus y)$	bitwise XOR inverted	8	13.6
`_or`	$(\neg x) \vee y$	not and bitwise OR	16	13.3
`inv`	$\neg x$	inversion	8	5.3
`div2`	$x >> 1$	division by 2	0	0.0
`div4`	$x >> 2$	division by 4	0	0.0
`add`	$x + y$	add	38	49.3
`adds`	$\min(x + y, 255)$	add with saturation	45	55.7
`mean`	$(x + y) >> 1$	average	40	50.3
`max`	$\max(x, y)$	maximum	61	60.0
`min`	$\min(x, y)$	minimum	61	60.0

usually completely different, the final cost (area) is expected to be roughly a sum of the areas required for both filters

IV. CIRCUIT EVOLUTION USING CGP

Functional-level as well as gate-level evolution of multifunctional circuits will be performed by CGP which has been applied for evolution and optimization of combinational circuits (i.e., acyclic directed graphs) for more than 10 years [19], [20], [21].

To model a generic combinational circuit, a candidate solution is represented as an array of n_c (columns) $\times$ n_r (rows) of 2-input processing elements. All candidate circuits have n_i primary inputs and n_o primary outputs. Every processing element can be connected either to the output of an element placed in previous L columns or to one of the primary inputs. A processing element can perform either a single function (then it is not a multifunctional element) or two functions. In the second case, the processing element is considered as multifunctional providing that the first function is activated in the first mode and the second function is activated in the second mode. The set of available functions is denoted Γ. Table I shows typical 8-bit functions for evolution of image filters.

The chromosome is a list of integers starting with the value of c if a constant-outputting function is included to Γ. Then, it contains $n_c \times n_r$ triplets, each of them encoding a single processing element (input1, input2, function code). The primary inputs are addressed by $0, 1 \ldots n_i - 1$ and processing elements by $n_i \ldots n_c n_r + n_o - 1$. The last n_o integers define the primary outputs of the circuit. An example of chromosome and a corresponding circuit is given in Fig. 5. Polymorphic control is modeled by Boolean signal s. In the first mode ($s = 0$), the functions shown in the upper part of boxes are active. In the second mode ($s = 1$), the functions of the bottom part are active. Notice that some elements (13 and 16) are not utilized.

CGP usually employs a simple $(1 + \lambda)$ evolution strategy to search in the space of candidate circuits [19]. The initial population is randomly generated or seeded using conventional designs. Then, it is evaluated and the best-scored individual is considered as the parent for a new population. However, as a new parent an offspring is always chosen if it is equally as fit or has better fitness than the parent. CGP uses a mutation operator to create λ offspring of the parent to fill the new population. The mutation randomly picks μ integers and replaces them by randomly generated (but legal) values. The evolution is terminated after producing g generations.

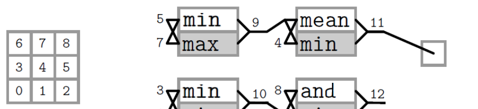

Fig. 5. Function-level CGP supporting multifunctional elements: $n_i = 9$, $n_o = 1$, $n_c = 2$, $n_r = 2$, $L = 2$, $R = \{min(0), max(1), mean(2), min/max(3), mean/min(4), and/min(5)\}$. Chromosome: 5,7,**3**; 3,1,**0**; 9,4,**4**; 8,10, **5**; 11. Function codes are typed bold.

Fitness function is application specific. In some cases, all possible input assignments are generated and resulting vectors are compared with requested vectors. The fitness value is then the number of correctly calculated bits. In other cases, only a subset of input vectors is utilized or specific procedures such as simulation of electronic properties [22] or functional equivalence checking [20] are applied. Multifunctional circuits have to be evaluated in all supported modes of operation.

We will present particular setting of CGP parameters and application-specific fitness functions in Sections V and VI.

V. EVOLUTION OF MULTI-FUNCTION FILTERS AT RTL

A. CGP Setting and Fitness Function

All candidate circuits have nine 8-bit primary inputs and one 8-bit primary output, i.e. $n_i = 9, n_o = 1$. Processing elements accept two 8-bit inputs and provide a single 8-bit output. Multifunctional elements are composed of two functions taken from Table I. A subset of functions used in a particular experiment will be denoted R. Figure 5 shows an example of a candidate circuit and its encoding.

Similarly to evolution of single-purpose filters, the original uncorrupted (training) image is needed to determine the fitness value. The goal of CGP is to find a circuit minimizing the difference between the original image and the output of the filter in both modes. The quality of filtering is expressed as the mean absolute error per pixel (*mdpp*):

$$f = \frac{1}{2N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} (|B_1(i,j) - C_1(i,j)| + |B_2(i,j) - C_2(i,j)|). \quad (1)$$

The ideal image which we are attempting to reach in the first mode (second mode) is denoted by C_1 (C_2). The filtered image which was obtained in the first mode (second mode) is denoted by B_1 (B_2). Finally, $N \times N$ denotes the size of image. Figure 6 shows the ideal version of the training image. Note that the functionality is the only objective applied in this

Fig. 6. Training image, 256×256 pixels

TABLE II
DILATATION/EROSION – COMPARISON OF SIX CGP CONFIGURATIONS.

CGP array size	4×4		5×5		6×6	
L	2	3	2	3	2	3
Average $mdpp$	3.44	**3.37**	4.62	4.59	5.03	4.75
Best $mdpp$	**0.25**	0.68	1.18	1.64	1.64	1.60

paper. The circuit cost is controlled only indirectly by setting the maximum number of processing elements ($n_r \cdot n_c$).

Experiments were performed for three target multifunctional filters. We started with the following setting of CGP parameters: $\lambda = 4$; $\mu = 1$ integer; the number of generations g =20,000; the number of independent runs $r = 40$; all possible functions and their combinations from Table I were allowed in R. We compared $mdpp$ for different setting of n_c, n_r and L. In the following subsections, results are given for the training image.

B. Case study 1: Dilatation/Erosion

In order to test whether the basic version of the proposed method works, we firstly evolved a simple dilatation/erosion filter. It is a very good candidate for multifunctional approach since an ideal dilatation filter calculates the maximum out of the input pixels and similarly, an ideal erosion filter calculates the minimum out of the input pixels [18]. Figure 7 shows the best evolved filter which is very close to the expected solution. Table II gives the best $mdpp$ (the best run) and the average $mdpp$ (the average out of 40 runs) in both modes for six different CGP configurations. The best solution was obtained for the smallest CGP array.

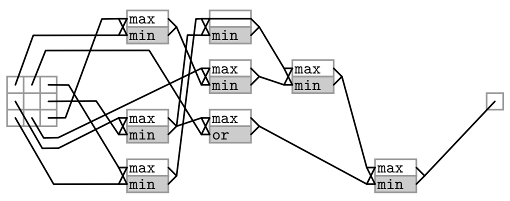

Fig. 7. A filter evolved for Dilatation/Erosion

TABLE III
EDGES/SHOTS – COMPARISON OF SIX CGP CONFIGURATIONS.

CGP array size	4×4		5×5		6×6	
L	2	3	2	3	2	3
Average $mdpp$	**8.29**	9.53	9.47	8.63	9.00	9.14
Best $mdpp$	6.39	6.49	6.01	5.54	**5.00**	5.28

TABLE IV
EDGES/SHOTS – THE EFFECT OF MUTATION AND FUNCTION SET SELECTION.

CGP array size	6×6					
L	2					
Max. poly functions	10			unlimited		
Mutations	1	2	3	1	2	3
Average $mdpp$	7.13	6.51	**5.71**	8.06	7.13	7.13
Best $mdpp$	4.53	4.79	**3.47**	5.30	4.73	3.94

C. Case study 2: Edges/Shots

The second objective was to evolve a filter performing edge detection in the first mode and salt and pepper noise (5%) elimination in the second mode (the Edges/Shots filter, for short). In order to create a training image for edge detection, we applied Sobel detector on the ideal image and used the resulting image as the target for evolution. As the edge detector (see e.g. [18]) and shot noise filter (Fig. 3) have very different structures, we expected that obtained circuits will be more complex than the dilatation/erosion filter.

We can see from Table III that while the best $mdpp$ was obtained for the largest CGP array, the average $mdpp$ is minimal for the smallest CGP array. Hence we performed additional experiments with the 6×6 array where we modified the mutation rate, reduced the function set and increased the number of generations. Table IV shows that significantly better results were obtained for the reduced function set and higher mutation rate. The best-performing filter is shown in Figure 8. Its performance is demonstrated in Fig 9.

D. Case study 3: Gauss/Shots

The third multifunctional filter we evolved is capable of suppressing the Gaussian noise ($\sigma = 0.1$ for normalized inputs $\langle 0{,}0; 1{,}0 \rangle$) in the first mode and the shot noise (the 5% salt and pepper noise) in the second mode (the Gauss/Shots filter, in short). On the basis of previous experiments, we modified the

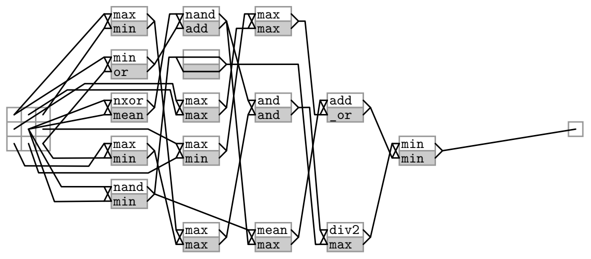

Fig. 8. A filter evolved for Edges/Shots noise

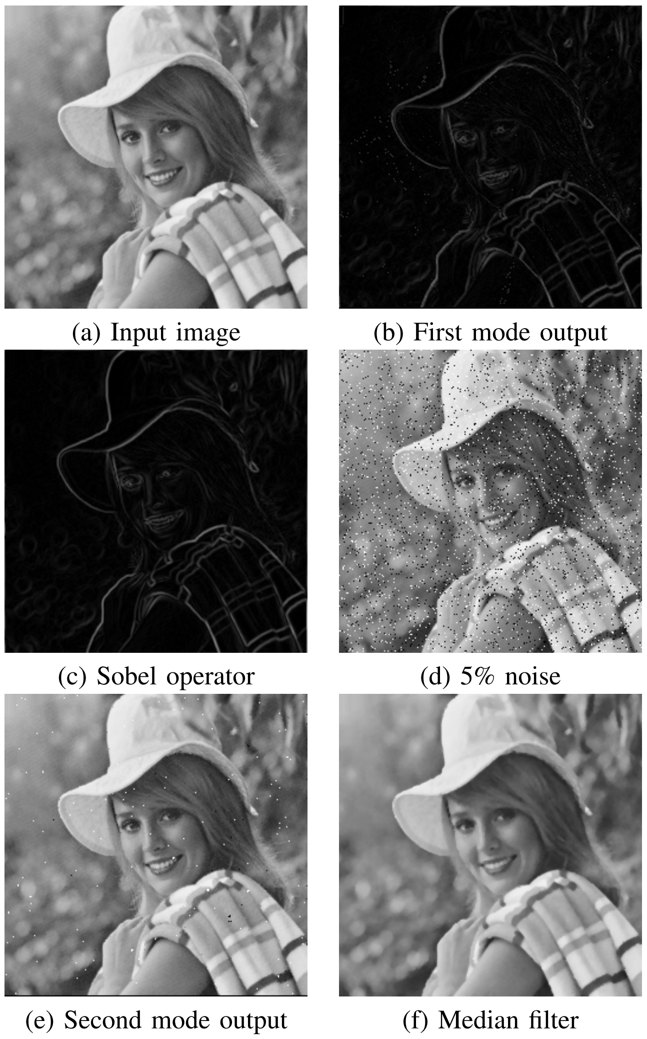

Fig. 9. Edges/Shots – results for both modes of the multifunctional filter and conventional filters.

TABLE V
GAUSS/SHOTS – COMPARISON OF SIX CGP CONFIGURATIONS.

CGP array size	4×4		5×5		6×6	
L	2	3	2	3	2	3
Average *mdpp*	12.60	10.94	12.67	11.33	13.96	**10.85**
Best *mdpp*	8.20	7.68	8.85	7.56	8.85	**7.40**

CGP setting to $\mu = 2$, $r = 20$, $g = 40000$. A comparison of results obtained for six CGP configurations is given in Table V. Additional results reported in Table VI indicate that restricted function set, $L = 4$, $n_c = 7$ and $n_r = 4$ give the best minimum as well as the average *mdpp*.

The best-evolved filter is shown in Fig. 10. While the mean function is the most frequent one used in the first mode, the functionality of the second mode is based on computing the minimum and maximum. We expected this type of function utilization.

TABLE VI
GAUSS/SHOTS – EXTENDED TESTING

CGP array size	7×4		8×5	
L	4		5	
Max. poly functions	8	unlimited	8	unlimited
Mutations	2	2	2	2
Average *mdpp*	**8.01**	12.14	8.72	13.35
Best *mdpp*	**6.40**	7.32	6.63	7.39

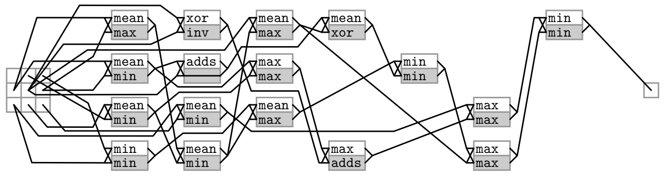

Fig. 10. A filter evolved for Gauss/Shots noise

E. Filtering Quality and Design Time

1) Filtering Quality: The best filters that we presented in previous subsections were compared in both modes with conventional filters. Table VII gives the average *mdpp* obtained using a set of 16 test images shown in [8]. The evolved salt-and-pepper filters exhibit lower *mdpp* with respect to the median filter (Fig. 3). The average *mdpp* of the evolved filter for Gaussian noise is slightly worse than the *mddp* obtained for the conventional averaging filter. Results are not given for dilatation, erosion and edge detection because corresponding conventional operators were directly utilized to create training images (i.e. *mdpp* is 0.0).

2) Time of Evolution: We measured the time of evolution for the dilatation/erosion filter ($\lambda = 4$, $g = 500$, $\mu = 1$, $L = 2$, $n_c = n_r = 4$) using a laptop equipped with the Pentium M 1.8 GHz processor. Figure 11 shows that the time of evolution significantly depends on the training image size. A typical experiment utilizing a 256×256-pixel image and running for 20,000 generations then took approx. 48 min.

VI. Gate-Level Optimization of Processing Elements for Multi-Function Filters

In this section, we will explore possible gate-level implementations of the 16-bit input (2 x 8 bits) and 8-bit output processing elements (such as min/max) that represent building blocks of evolved filters shown in Fig. 7, 8 and 10. Resulting implementations of evolved filters will be then compared with conventional solutions based on multiplexing of the common

TABLE VII
COMPARISON OF MDPP FOR EVOLVED FILTERS AND CONVENTIONAL FILTERS USING TEST IMAGES

	evolved filter	conventional filter
Dilatation/Erosion	0.1306 / 0.4160	– / –
Edges/Shots	4.3896 / 2.2172	– / 4.2166
Gauss/Shots	10.3673 / 2.2676	10.0192 / 4.2166

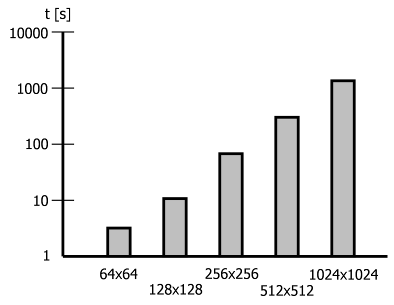

Fig. 11. The evolution time w.r.t. the training image size for $\lambda = 4$, $g = 500$, $\mu = 1$, $L = 2$, $n_c = n_r = 4$.

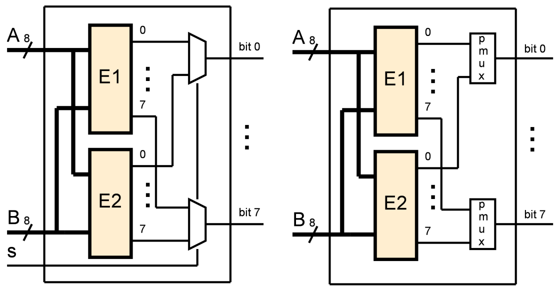

Fig. 12. Implementations of a processing element: multiplexer-based (left) and polymorhic multiplexer-based (right). E1 and E2 are elementary functions from Table I.

filters (e.g. median filter and Sobel filter) as presented in Section III.

A. Multiplexer-based method

Figure 12 (left) shows the multiplexer-based approach proposed to implement the multifunctional processing elements. The construction has three steps.

(1) Elementary functions from Table I are described in VHDL and synthesized using LeonardoSpectrum (LS) which can utilize the gates from Table VIII (except NAND/NOR). The implementation cost of resulting gate-level circuits is expressed as a relative area and in absolute gate numbers in Table I.

TABLE VIII
RELATIVE IMPLEMENTATION COST (AREA) OF GATES

Area	Gate
0.67	NOT
1.00	NAND, NOR
1.33	AND, OR
1.66	XNOR
2.00	XOR
2.00	NAND/NOR

(2) Two elementary functions (E1 and E2 in Fig. 12) are then connected using standard multiplexers in order to obtain a single multifunctional processing element controlled by selector s.

(3) Implementations of evolved filters (Fig. 7, 8 and 10) are composed of the processing elements created in step (2). A particular filter selection depends on setting of the selector s.

TABLE IX
THE COST (AREA) OF FILTERS CREATED USING VARIOUS APPROACHES

Evolved filter	CoMux	CoMux ABC	Polymux	Polymux ABC	Polymux ABC-CGP
Dilatatation/Erosion	880	569	1277	1048	659
Edges/Shots	1343	1089	1614	1458	1156
Gauss/Shots	1857	1639	2015	1950	1735

The final implementation cost of the filters is given by the CoMux column in Table IX.

The gate-level netlists of processing elements can further be optimized, e.g. by conventional tools such as ABC [23]. The cost of filters composed of the processing elements that were optimized using ABC is given by the CoMux-ABC column in Table IX.

B. Polymorphic multiplexer-based method

The idea of polymorphic multiplexing of elementary functions in the processing element is shown is Fig. 12 (right). The first step of construction – the gate level design of elementary functions – is the same as in the previous procedure. Then two elementary functions (E1 and E2) are connected using polymorphic multiplexers in order to obtain a single multifunctional processing element. Because every polymorphic multiplexer contains two polymorphic gates (Fig. 2) each multifunctional processing element contains 16 polymorphic gates. The application of this basic construction procedure leads to multifunctional filters whose implementation cost is given by the Polymux column in Table IX.

The gate-level netlists of the processing elements can further be optimized. However, as neither ABC nor LS support polymorphic gates, we can optimize only the implementations of interconnected elementary functions E1 and E2 in the processing element. Table X (the seed columns), shows the number of gates (including polymorphic gates) in the processing elements obtained after an optimization conducted using ABC and LS respectively. It can be seen that ABC and LS provides very similar results in average. The optimized implementations of the processing elements were utilized as building blocks for evolved filters. Table IX, the Polymux-ABC column, shows that a small improvement has been obtained in all three cases w.r.t. the Polymux approach.

C. Postsynthesis optimization using CGP

In order to further optimize the multifunctional processing elements, we considered the resulting processing elements that were obtained in the previous section as a seed for the CGP circuit optimizer. The CGP array contains $n_c \times 1$ nodes, where n_c is the number of gates in the seed circuit, $n_i = 16$, $n_o = 8$, $\lambda = 1$, $l = n_c$ and $\mu = 2$. The values of parameters are chosen according to [6]. The set of functions includes ordinary two-input logic functions, buffer, inverter and the NAND/NOR gate, i.e. $\Gamma = \{\mathrm{BUF}, \mathrm{NOT}, \mathrm{AND}, \mathrm{OR}, \mathrm{XOR}, \mathrm{NAND}, \mathrm{NOR}, \mathrm{NAND/NOR}, \mathrm{ZERO}, \mathrm{ONE}\}$. Similarly to [5], we have used just one polymorphic function NAND/NOR.

1) Fitness Function: A straightforward approach to the evaluation of a candidate polymorphic combinational circuit requires applying 2^{n_i} assignments to the inputs, calculating the number of correctly produced bits for the first mode and repeating these two steps for the second mode. This is very time consuming for our target 16-input/8-output multifunctional circuits. In order to accelerate the evaluation, a SAT-based equivalence checking algorithm is applied [6]. This algorithm assumes that CGP is seeded using a fully functional polymorphic circuit. The seed is utilized as a *reference* solution for the SAT-based equivalence checking algorithm.

The fitness evaluation works as follows. The reference circuit U as well as candidate circuit V (which is created by mutation from the parent) is set to the first mode. A new auxiliary circuit W_1 is composed of the reference circuit in the first mode (circuit U_1), the candidate circuit in the first mode (circuit V_1) and a miter (a set of XOR gates followed by the OR detector), see Fig. 13. Circuit W_1 is transformed into one Boolean formula in conjunctive normal form (CNF) which is unsatisfiable if and only if circuits U_1 and V_1 are functionally equivalent [24]. The transformation to CNF is conducted gate by gate using the Tseitin's algorithm [25]. If U_1 and V_1 are not functionally equivalent then the fitness evaluation is finished and CGP proceeds with another candidate circuit. Otherwise, circuits U and V are set to the second mode and the process is repeated. If the circuits are also functionally equivalent in the second mode then the fitness value is given by the number of gates. Otherwise, the fitness value is -1, i.e. the worst one.

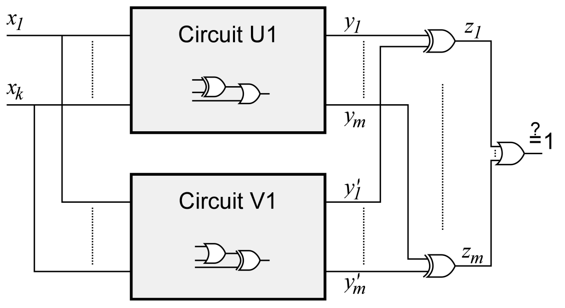

Fig. 13. Equivalence checking of two combinational circuits

2) Results: The MiniSAT 2 (version 070721) has been used as a SAT solver [26] because it can easily be embedded into a custom application. The experiments were carried out on a cluster consisting of Intel Xeon E5345 2.33 GHz processors that enables to run several experiments in parallel. Two seeds were compared – one coming form the ABC tool and another one from the LS tool. For each seed and processing element, 25 parallel runs of CGP were performed. Every 15 minutes, all runs were re-seeded using the best individual obtained so far. The evolution was stopped when no progress has been observed in last 15 minutes. The minimum, average and maximum time spend by a single run was 1.25, 1.61 and 4 hours.

Table X shows the number of gates in the best circuit, the number of polymorphic gates and resulting relative area. We can observe that the averages calculated for our set of processing elements depend on the seed chosen insignificantly.

As example, Figure 14 shows the progress of optimization of the nand/min processing element seeded using a circuit comming from the ABC tool.

The most area-efficient implementations of processing elements were then utilized to create the image filters according to Fig. 7, 8 and 10. The Polymux ABC-CGP column in Table IX shows a significant reduction in the area in comparison with the Polymux-ABC method.

TABLE X
IMPLEMENTATION COST OF PROCESSING ELEMENTS BASED ON POLYMORPHIC MULTIPLEXING, OPTIMIZED BY CGP WHICH WAS SEEDED EITHER BY ABC OR LS.

processing	**Seeded by ABC**				**Seeded by LS**			
	seed	**best**			**seed**	**best**		
element	**gates**	**gates**	**polygates**	**area**	**gates**	**gates**	**polygates**	**area**
max / add	134	92	22	139.2	139	99	22	151.0
max / div2	102	67	16	98.9	102	68	18	103.3
mean / max	138	90	28	135.9	141	94	24	141.8
min / max	123	52	22	86.3	162	52	22	88.6
min / mean	146	97	27	148.8	141	96	21	144.9
nand / and	56	17	1	26.2	56	17	1	26.1
nand / min	118	65	19	100.9	109	61	16	100.0
_or / adds	100	57	21	93.8	101	58	22	99.5
or / max	112	60	14	91.9	109	52	12	90.3
ident / add	76	53	16	88.1	78	53	14	90.1
ident / min	101	46	4	68.3	101	47	6	70.7
nxor / mean	79	60	19	102.9	88	59	19	97.8
xor / inv	56	18	1	26.8	56	17	1	25.5
xor / mean	85	56	13	83.5	88	53	11	83.0
Average	101.9	59.2	15.9	92.3	105.1	59.5	14.9	93.8

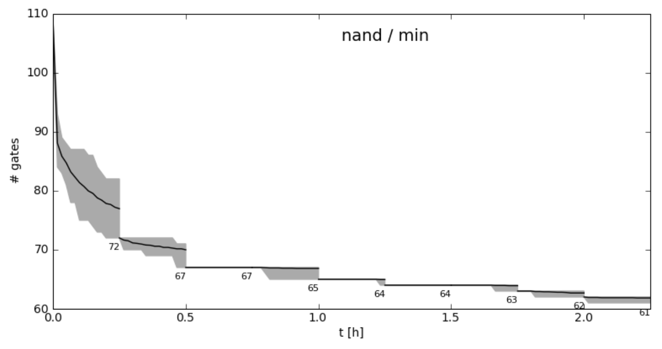

Fig. 14. The minimum, maximum and average number of gates from 25 runs of CGP for the nand/min processing element

D. Comparison with conventional filters

As proposed in Section III, the conventional implementations of filters can be multiplexed to develop multifunctional filters. Hence we synthesized gate-level implementations of the 9-input median filter, 9-input averaging filter (9-Mean), 9-input dilatation and 9-input erosion filter. Subsequently, multifunctional filters were composed using multiplexers. Note that the total area of the 8-bit multiplexer is 10.64. Table XI gives the implementation cost of conventional filters and multiplexed conventional filters.

By comparison of the relative costs, it can be seen that evolved filters that were implemented using multiplexing at the level of processing elements (Table IX) occupy much smaller area than multiplexed conventional filters (Table XI). If the CGP optimization is taken into account, polymorphic multiplexing-based implementations are comparable to implementations utilizing multiplexed processing elements. It is important to emphasize here that the implementation cost of polymorphic multiplexers is considered as relatively high and more compact implementations are expected in the future.

TABLE XI
IMPLEMENTATION COST OF CONVENTIONAL 9-INPUT OPERATORS AND FILTERS OBTAINED BY MULTIPLEXING OF CONVENTIONAL OPERATORS.

	components						total	
operator	**min**	**max**	**mean**	**add**	**div2**	**mux**	**gates**	**area**
Mean			8				320	402.4
Median	15	15					1830	1800.0
Dilatation		8					488	480.0
Erosion	8						488	480.0
Sobel	1	1		5	2		312	366.5

	components						total	
filter	**min**	**max**	**mean**	**add**	**div2**	**mux**	**gates**	**area**
Dilatation / erosion	8	8				1	977	970.6
Mean / median	15	15	8			1	2151	2213.0
Sobel / median	16	16		5	2	1	2143	2177.1

VII. CONCLUSIONS

A new two-step method for design of multifunctional image filters has been proposed in this paper. In the first step, we applied CGP to evolve image filters composed of multifunctional processing elements operating over 8 bits. This approach extended our previous work on image filter evolution. It was shown that CGP can fit two different filtering tasks to a single acyclic directed graph and the resulting filters exhibit desired filtering quality. In the second step, we tested several methods to implement processing elements involved in evolved filters in order to minimize the area on a chip. Simple multiplexing proved to be a very efficient method to accomplish this task. By combining of CGP with SAT-based fitness function, we were able to find interesting solutions for circuits with polymorphic gates even when a very pessimistic implementation cost was assumed for polymorphic multiplexers.

VIII. ACKNOWLEDGMENTS

This work was supported by the Czech science foundation project P103/10/1517, the research programme MSM 0021630528, the BUT project FIT-S-11-1 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

REFERENCES

[1] S. Hauck and A. DeHon, *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation*. Morgan Kaufmann, 2007.

[2] S. Tanachutiwat, J. U. Lee, W. Wang, and C. Y. Sung, "Reconfigurable multi-function logic based on graphene p-n junctions," in *Design Automation Conference, DAC*. ACM, 2010, pp. 883–888.

[3] 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.

[4] 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.

[5] Z. Gajda and L. Sekanina, "On evolutionary synthesis of compact polymorphic combinational circuits," *Journal of Multiple-Valued Logic and Soft Computing*, vol. 17, no. 6, pp. 607–631, 2011.

[6] L. Sekanina and Z. Vasicek, "A sat-based fitness function for evolutionary optimization of polymorphic circuits," in *Proc. of the Design, Automation and Test in Europe, DATE.* EDAA, 2012, pp. 715–720.

[7] Z. Vasicek and L. Sekanina, "An area-efficient alternative to adaptive median filtering in fpgas," in *Proc. of the 17th Conf. on Field Programmable Logic and Applications*. IEEE CS, 2007, pp. 216–221.

[8] S. Harding and W. Banzhaf, "Genetic programming on gpus for image processing," in *Proc. of the First Int. Workshop on Parallel and Bioinspired Algorithms*. Complutense University of Madrid Press, 2008, pp. 65 – 72.

[9] J. Torresen, "Two-step incremental evolution of a prosthetic hand controller based on digital logic gates," in *Proc. of the 4th Conf. on Evolvable Systems: From Biology to Hardware*, 2001.

[10] L. Sekanina and V. Salajka, "Towards new applications of multi-function logic: Image multi-filtering," in *Proc. of the Design, Automation and Test in Europe, DATE.* EDAA, 2012, pp. 824–827.

[11] 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, pp. 39–46.

[12] 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.

[13] Z. Li, W. Luo, L. Yue, and X. Wang, "On the completeness of the polymorphic gate set," *ACM Transactions on Design Automation of Electronic Systems*, vol. 15, no. 4, p. 25, 2011.

[14] Z. Gajda and L. Sekanina, "Gate-level optimization of polymorphic circuits using cartesian genetic programming," in *IEEE Congress on Evolutionary Computation*. IEEE Computational Intelligence Society, 2009, pp. 1599–1604.

[15] H. Liang, W. Luo, and X. Wang, "Designing polymorphic circuits with evolutionary algorithm based on weighted sum method," in *Evolvable Systems: From Biology to Hardware*, ser. LNCS, vol. 4684. Springer, 2007, pp. 331–342.

[16] L. Sekanina, R. Ruzicka, and Z. Gajda, "Polymorphic FIR filters with backup mode enabling power savings," in *2009 NASA/ESA Conference on Adaptive Hardware and Systems*. IEEE, 2009, pp. 43–50.

[17] J. I. Smith, "Implementing median filters in xc4000e fpgas," *Xilinx Xcell*, vol. 23, p. 16, 1996.

[18] H. R. Myler and A. R. Weeks, *The Pocket Handbook of Image Processing Algorithms in C*. Prentice Hall, Inc., 1993.

[19] J. F. Miller, D. Job, and V. K. 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] Z. Vasicek and L. Sekanina, "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 305–327, 2011.

[21] J. F. Miller, *Cartesian Genetic Programming*. Springer-Verlag, 2011.

[22] J. A. Walker, J. A. Hilder, D. Reid, A. Asenov, S. Roy, C. Millar, and A. M. Tyrrell, "The evolution of standard cell libraries for future technology nodes," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 235–256, 2011.

[23] Berkley Logic Synthesis and Verification Group, *ABC: A System for Sequential Synthesis and verification*, 2010. [Online]. Available: http://www.eecs.berkeley.edu/~ alanmi/abc/

[24] E. Goldberg, M. Prasad, and R. Brayton, "Using SAT for combinational equivalence checking," in *DATE '01: Proceedings of the conference on Design, automation and test in Europe*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 114–121.

[25] G. S. Tseitin, "On the complexity of derivation in propositional calculus," in *Studies in Constructive Mathematics and Mathematical Logic, Part II*, 1968, pp. 115–125.

[26] N. Een and N. Sorensson, "MiniSAT." [Online]. Available: http://minisat.se