Evolutionary Design of Approximate Multipliers Under Different Error Metrics

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

***Abstract*—Approximate circuits are digital circuits which are intentionally designed in such a way that the specification is not met in terms of functionality in order to obtain some improvements in power consumption, performance or area, in comparison with fully functional circuits. In this paper, we propose to design approximate circuits using evolutionary design techniques. In particular, different error metrics are utilized to assess the circuit functionality. The proposed method begins with a fully functional circuit which is then intentionally degraded by Cartesian genetic programming (CGP) to obtain a circuit with a predefined error. In the second phase, CGP is used to minimize the number of gates or another error criterion. The effect of various error metrics on the search performance, area and power consumption is evaluated in the task of multiplier design.**

I. Introduction

The functional equivalence between the *specification* and *implementation* has always been the first and crucial attribute of any *digital circuit*. Then the most suitable trade off has been sought among the secondary (from this point of view) circuit parameters such as area, delay and power consumption. Power consumption is currently considered as the most important circuit parameter in many applications. Hence various approaches to power reduction are applied at all design levels, starting from the architecture via the circuit to the technology [1]. It seems that further reductions can only be obtained by approximating the original circuit function, which is the goal of a nascent field of *approximate computing*. Another motivation for approximate computing can be seen in the recently introduced concept of underdesigned and opportunistic computing which attempts to explore the possibility of constructing machines naturally exploiting various imperfections of hardware and capability of some applications and users to tolerate imperfections in computing [2].

The digital circuits which are intentionally designed in such a way that the specification is not met in terms of functionality and some savings are expected in terms of energy, delay or area are called *approximate circuits* [3], [4], [5]. They are suitable for those applications in which errors are not recognizable as human perception capabilities are limited (e.g. in multimedia applications), no golden solution is available for validation of circuit behavior (e.g. in data mining applications), or users are simply willing to accept some inaccuracies (e.g. when battery of a mobile phone is almost depleted).

After presenting several approximate circuits that were created manually [9], designers currently try to develop general purpose design methods for approximate circuits. All currently available methods are *error-oriented* in the sense that all logic optimizations leading to an approximate solution are constrained by a predefined *error criterion* [6], [3], [7]. The error can be expressed by various metrics such as the error magnitude, average error magnitude or error probability. However, these methods do not allow to precisely tune the error. The design process has to be repeated when a new requirement on the error is specified.

In our previous work, we introduced an evolutionary gate-level design method based on Cartesian genetic programming (CGP) for purposes of approximate circuits construction [8]. We exploited the fact that power consumption is often highly correlated with occupied resources and the evolutionary design is capable of constructing partially working solutions even if sufficient resources (required for finding a fully functional solution) are not available. Supposing that n is the minimum number of gates required for a complete functionality, we let CGP minimize the error providing that only $n-1$ gates are available. The process can be repeated for $n-2$, $n-3$ etc. gates. The user thus obtains a set of approximate combinational circuits, each of which typically exhibits different trade off between the functionality, the number of gates and delay. This approach can be considered as an *area-oriented* method because the user can control the used area (and so power consumption) more comfortably than by means of the error-oriented methods.

In this paper, we propose a complementary evolutionary design approach to our original work [8]. The user is supposed to define a required error level e_{max} (e.g. the average error magnitude). CGP, which is seeded by a conventional fully functional implementation, is utilized to modify the seed in order to obtain a circuit with predefined e_{max}. After obtaining that circuit several scenarios can be taken. CGP can minimize the mean error or the number of gates providing that e_{max} is guaranteed. We will analyze these scenarios in order to deliver the most practically useful solutions. The proposed approach can be classified as error-oriented. The method will be evaluated in the task of a gate-level combinational multiplier design. Results will be compared with the conventional fully functional implementations of multipliers and the approximate multipliers that have been reported in the literature.

In summary, the key contributions of this article are as follows:

- We propose a new methodology for approximate circuit design which exploits the error-oriented evolutionary circuit design.
- We analyze the impact of different optimization scenarios on the parameters of evolved approximate circuits.
- We present novel implementations of approximate combinational multipliers.

The rest of the paper is organized as follows. Section II surveys the relevant research. After introducing CGP in Section III, the proposed optimization method is presented in Section III-C. Sections IV-A, IV-B and IV-C contain the experimental setup, results and discussion. Conclusions are given in Section V.

II. Related Work

A. Approximate Circuits

Approximate circuits can be constructed by means of voltage over-scaling and over-clocking. In this paper, we will solely employ the methods based on functional approximations in which circuits are designed in such a way that they do not fully follow the logic behavior prescribed by the specification. A good example is a two-bit multiplier which was manually constructed using 5 gates (with delay of 2 gates). Its output is correct for 15 out of 16 possible inputs. A usual conventional solution requires 8 gates and exhibits the delay of 3 gates. This approximate multiplier has been used in larger approximate multipliers and then employed in approximate image processing applications [9].

In order to automate the whole design process, the current trend is to create systematic design methodologies for approximate circuits. The Systematic methodology for Automatic Logic Synthesis of Approximate circuits (SALSA) starts with a RT level description of the exact version of the circuit and an error constraint that specifies the type and amount of error that the implementation can exhibit [6]. The methodology introduces the so-called Q-function which takes the outputs from both the original circuit and approximate circuit and decides if the quality constraints are satisfied. The Q-function outputs a single Boolean value. The SALSA algorithm attempts to modify the approximate circuit with the goal of keeping the output of the Q function unchanged.

Another systematic approach, Substitute-And-SIMplIfy (SASIMI), tries to identify signal pairs in the circuit that exhibit the same value with a high probability, and substitutes one for the other [7]. These substitutions introduce functional approximations. Unused logic can be eliminated from the circuit which results in area and power savings. The method is combined with technology-level optimizations such as downsizing of gates on critical paths and voltage scaling which results in additional significant area and power savings.

B. Evolutionary Circuit Design

The field of evolutionary circuit design has been surveyed in several papers, e.g. [10], [11], [12]. Some of the works can be considered as examples of approximate circuits: Miller evolved finite impulse response filters at the gate level, where functionality was traded for area [13], and Kneiper et al. investigated the robustness of evolved classifiers [14] in which available resources were changing over time. However, these approaches have not explicitly considered the approximate computing paradigm.

In our previous paper [8], we evolved approximate implementations of small combinational circuits (3-bit and 4-bit adders and single output circuits) using randomly seeded CGP operating at the gate level, where the goal was to minimize the error under some area constraints. An inherently *multiobjective approach* to evolutionary design of approximate multiplierless multiple constant multipliers (MCMs) was proposed in [15]. Three design objectives—accuracy, area and delay—were optimized by multiobjective CGP, where the area was inexpensively estimated as the number of utilized components and delay as the number of components along the longest path between the input and the output. In both cases the initial solutions were generated randomly.

C. Error Functions

To evaluate the quality of a particular w-bit approximate arithmetic circuit, several error criteria can be used [6], [7]. Let $O^{(i)}_{orig}$ be an output value of the fully functional circuit for an input vector i and $O^{(i)}_{approx}$ be an output value of an approximate circuit. Then, the following metrics can be calculated.

Error magnitude (e_{wst}), sometimes denoted as the *worst case error* and defined as

$$e_{wst} = \max_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}| \tag{1}$$

represents the fundamental metrics. If used as the optimization goal (i.e. the condition $e_{wst} \leq K$ is satisfied during the whole design process) this metric guarantees that the approximate output can differ from the correct output by at most K.

Relative error, which is defined as the ratio of the approximate and original values, and its maximal value (e_{rel}) defined as

$$e_{rel} = \max_{\forall i} \frac{|O^{(i)}_{orig} - O^{(i)}_{approx}|}{O^{(i)}_{orig}} \tag{2}$$

represents another criterion used to constrain the approximate circuit to differ from the original one by at most a certain margin. For a given constant K ($0 < K < 1$), the validity of the expression $e_{rel} \leq K$ must be ensured during the optimization. Note that a special care must be devoted to the cases for which the output value of the original circuit is equal to zero.

Average error magnitude (e_{avg}) is defined as the sum of absolute differences in magnitude between the original and approximate circuits (i.e. total error e_{tot}), averaged over all inputs:

$$e_{avg} = \frac{e_{tot}}{2^{2w}} = \frac{\sum_{\forall i} |O^{(i)}_{orig} - O^{(i)}_{approx}|}{2^{2w}} \tag{3}$$

Error probability (*error rate*) defined as the percentage of inputs vectors for which the approximate output differs from the original one represents the last commonly used metrics:

$$e_{prob} = \frac{\sum_{\forall i, O^{(i)}_{orig} \neq O^{(i)}_{approx}} 1}{2^{2w}} \quad (4)$$

However, this metrics is utilized mostly in conjunction with the design of an approximate version of a non-arithmetic digital circuit, i.e. for circuits that do not produce a single binary-encoded output value.

Instead of the absolute error values e_{wst} and e_{avg} depending on the utilized bit width w, the corresponding percentage error values $e_{wst\%}$ and $e_{avg\%}$ can be utilized. In this case, it is common to express the percentage error ratio as the percentage of the maximum value that can occur on the output of the original circuit:

$$e_{wst\%} = e_{max}/(2^w - 1)^2 \quad (5)$$

$$e_{avg\%} = e_{avg}/(2^w - 1)^2 \quad (6)$$

III. Evolutionary design of approximate multipliers

CGP and its various versions are probably the most popular methods for the evolutionary circuit design [16]. The principle of CGP and the proposed modification are described in the following paragraphs.

A. Circuit Representation

In general, CGP represents a candidate circuit by means of an array of processing nodes arranged in n_c columns and n_r rows.

Each circuit has n_i primary inputs and n_o primary outputs. Γ denotes the set of functions that can be executed by processing nodes. CGP utilizes the following encoding scheme. The primary inputs and processing node outputs are labeled $0, 1, \ldots, n_i - 1$ and $n_i, n_i + 1, \ldots, n_i + n_c.n_r - 1$, respectively. Each node input can be connected either to the output of a gate placed in previous l columns or to one of the primary circuit inputs. A candidate solution consisting of two-input nodes is represented in the chromosome by $n_c.n_r$ triplets (x_1, x_2, ψ) determining for each processing node its function $\psi \in \Gamma$, and label of nodes x_1 and x_2 which its inputs are connected to. The last part of the chromosome contains n_o integers specifying the labels of nodes or primary inputs where the primary outputs are connected to.

In order to encode the approximate circuits, we enable the primary outputs and the inputs of the processing nodes to be connected directly to logic constants '0' and '1'. The logic constants are encoded as two nodes labelled -1 and -2.

Figure 1 demonstrates the principle of CGP encoding. It can be seen that although 6 gates are available in total, not all the gates have to be employed in the resulting circuit. This support of redundancy represents one of the most important features of CGP encoding [16].

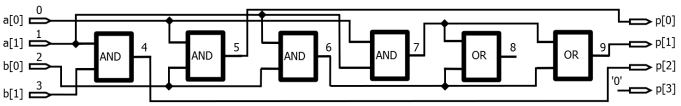

Fig. 1. A candidate 2-bit multiplier, with two inputs a and b and output p, represented by CGP with parameters: $n_i = n_o = 4$, $n_c = 6$, $n_r = 1$, $l = 4$, $\Gamma = \{0^{AND}, 1^{OR}\}$. Chromosome: 1, 3, 0; 0, 2, 0; 1, 2, 0; 0, 1, 0; 7, 6, 1; 7, 6, 1; 5, 9, 4, -1.

B. Search Strategy

As a search algorithm, CGP employs a $(1 + \lambda)$ evolutionary strategy [16]. This method consists of the following steps.

1) The initial population of the size $1 + \lambda$ is created.
2) The fitness function f is called for each candidate circuit.
3) The highest-scored candidate circuit is selected as the new parent. It has to be noted that the previous parent is never selected as the new parent if there exists at least one individual which obtained the same fitness value [16].
4) By applying a mutation operator, λ offspring individuals are generated from the parent.
5) Steps 2—4 are repeated until the termination condition is not satisfied.

CGP utilizes a point mutation which modifies up to h randomly chosen genes. The value of a selected gene is replaced with a randomly generated new one. Note that only a limited amount of values can be assigned to each gene. The range of valid values depends on the gene position and can be determined in advance. Crossover is not utilized.

C. Proposed method

In order to synthesize approximate w-bit multipliers with a required error magnitude and optimize the other circuit parameters (such as the area), the designer has to provide the required level of error (i.e. the constant K), the metrics that will be utilized to guarantee the chosen error level, circuit parameters to be optimized, and a fully functional w-bit multiplier.

Because we do not have a multiplier satisfying the given error criterion, the proposed method consists in two phases. The goal of the first phase is to evolve a multiplier which shows the error as close as possible to K. To achieve this objective, the fitness value f_{L1} is calculated according to the user requirement, i.e. according to the equation for error magnitude, average error magnitude or relative error. CGP is seeded by the fully functional multiplier which is gradually degraded by CGP until a circuit with the error K is obtained. CGP then continues by the second phase, in which the second optimization criterion f_{L2} is considered. It is guaranteed that the first optimization criterion is not worsening while the second criterion is simultaneously improving. The method is investigated in three scenarios which differ in the optimization objectives.

The aim of the *first scenario* (denoted S1) is to synthesize approximate multipliers showing the required error magnitude

K_{wst} and simultaneously having a minimal possible average error. To achieve this goal, the fitness value f_{L1} is calculated according to equation 1, while the second fitness value f_{L2} is calculated according to equation 3.

The *second scenario* (S2) is similar to S1, however the goal is to obtain a solution which occupies the minimum number of gates for a given error magnitude K_{wst}. In this case, the fitness value f_{L1} is calculated according to equation 1 and the fitness value f_{L2} is calculated as the number of gates in a candidate multiplier.

The goal of the *third scenario* (S3) is to evolve approximate multipliers which utilize the minimum number of gates for a certain average error magnitude K_{avg}. The fitness value f_{L1} is calculated according to equation 3 and the fitness value f_{L2} is calculated as the number of gates in a candidate multiplier.

Note that the evolutionary optimizer is implemented in such a way that a 5% deviance from the required average error magnitude is tolerated. This kind of flexibility is introduced because for some multipliers it is not possible to obtain an implementation which has exactly the specified average error magnitude.

IV. Experimental results and discussion

A. Experimental setup

The goal is to obtain an approximate version of a w-bit multiplier ($4 \leq w \leq 8$) if a fully functional solution and the required error level K are given. According to the results published in paper [6], [7], the error level was chosen to be 10% for K_{wst} and 0.5% for K_{avg}. The setting of CGP parameters, which is valid for all strategies, is as follows: Γ = {BUF, INV, AND, OR, XOR, NAND, NOR, XNOR}, $\lambda = 5$, $h = 1$, $l = n_c = N_g$, $n_r = 1$, $n_i = n_o = 2w$, where N_g is the number of gates of the fully functional w-bit multiplier. The values of these parameters are chosen according to our previous study [8]. The parameters of the fully functional multipliers (constructed as the ripple carry array multipliers) which will be serving as the reference circuits in the next comparisons are given in Table I.

TABLE I
Parameters of the original fully functional multipliers

w	N_g	power [uW]	area	delay [ns]
2	8	53.9	12.0	7.5
3	30	189.3	45.9	28.6
4	64	550.7	98.5	46.4
5	110	1508.9	169.8	64.2
6	168	3185.1	259.6	82.0
7	238	5958.2	368.1	99.8
8	320	10204.7	495.3	117.6

We utilized SIS software [17] to calculate the power consumption, occupied area and delay. The calculations are valid for the MCNC library [17], $V_{dd} = 5$ V and 20 MHz. The relative area of the used gates is: BUF 0.0, INV 0.67, NAND and NOR 1.00, AND and OR 1.33, XOR 2.00, XNOR 1.66. The sum of relative areas of gates used in a particular circuit will be denoted 'area'. It has to be noted that the utilized technology is obsolete, however we would like to demonstrate the relative power consumption with respect to the fully functional multiplier.

In order to accelerate CGP, we adopted an approach that have been proposed in [18], and implemented additional optimizations into CGP including the skipping of evaluation of neutral mutations and a premature termination of the fitness computation for unacceptable candidate solutions. The preliminary experiments showed that the number of candidate solutions evaluated within a time period increased more than 30 times using these changes. The experiments were carried out on a cluster consisting of 64-bit Intel Xeon processors running at 2.4 GHz. The evolutionary optimizer is terminated when a predefined number of generations g_{max} is exhausted. The parameter g_{max} was chosen such that the time of the evolutionary optimization of an 8-bit multiplier is less than 1.5 hour. For each scenario and given error-level, 50 independent evolutionary runs were performed.

B. Evaluation of the proposed methods

In order to evaluate the behavior of the proposed scenarios, three parameters calculated from all experimental runs are investigated. (1) The mean number of generations that the evolutionary optimizer spends in the first level of optimization. This parameter, denoted as $gens_{L1}$, gives an insight to the complexity of the evolutionary design of an approximate version of the original multiplier that exhibits the required error level K_{wst} (valid for S1 and S2) or K_{avg} (valid for S3). (2) The mean relative improvement of the occupied area $area_{L1}$ which indicates a percentage change of the number of utilized gates with respect to the original multiplier, and (3) $area_{L2}$ expressing a change against the candidate solution produced by the first level of optimization. The results are summarized in Table II. Note that only some error levels are shown due to the limited space.

According to the values of $gens_{L1}$, it is evident that the design of an approximate multiplier exhibiting the required error level represents a problem that can be relatively easily solved. No more than 200 thousand generations in average are required to accomplish this task. The required candidate solution can thus be produced within a few minutes of an evolutionary run. Nevertheless, scenario S3 requires two orders of magnitude higher number of generations in contrast with scenarios S1 and S2. This indicates that the discovering of an approximate circuit with a certain average error magnitude represents a more difficult problem than the design of an approximate circuit with a certain error magnitude.

The positive values of $area_{L1}$ indicate that the number of utilized gates is implicitly reduced with the increasing amount of erroneous responses of a candidate circuit. In all three cases, the circuits obtained after the first optimization step exhibit a small improvement of the utilized area. However, a different situation occurs during the second phase of the optimization. While S2 and S3 continue in improving of the occupied area, strategy S1 does not lead to any consequent power reduction. The reason of this behavior is probably caused by the fact

TABLE II
EVALUATION OF THE PROPOSED METHODS FOR SOME ERROR LEVELS

			S1			S2					S3		
w	K_{wst}	$K_{wst\%}$	$gens_{L1}$	$area_{L1}$	$area_{L2}$	$gens_{L1}$	$area_{L1}$	$area_{L2}$	K_{avg}	$K_{avg\%}$	$gens_{L1} \times 10^3$	$area_{L1}$	$area_{L2}$
4	6	2.7%	55 ± 79	1.3%	1.1%	29 ± 30	1.8%	28.4%	85	0.1%	115 ± 31	4.8%	2.0%
	12	5.3%	33 ± 52	0.9%	-0.5%	22 ± 23	1.9%	44.0%	155	0.3%	112 ± 22	7.9%	6.4%
	23	10.2%	38 ± 32	5.4%	-5.9%	34 ± 35	4.6%	64.6%	281	0.5%	102 ± 6	12.1%	9.7%
5	25	2.6%	46 ± 39	1.9%	-1.7%	42 ± 40	2.3%	35.9%	1231	0.1%	119 ± 28	6.0%	7.1%
	49	5.1%	52 ± 38	3.4%	-3.3%	44 ± 38	3.2%	54.9%	2461	0.3%	116 ± 24	7.6%	13.9%
	93	9.7%	54 ± 32	5.5%	-5.7%	61 ± 42	6.6%	72.6%	4675	0.5%	115 ± 23	10.8%	21.6%
6	96	2.4%	27 ± 33	0.8%	1.3%	44 ± 113	1.1%	46.3%	24385	0.1%	102 ± 4	7.2%	17.9%
	191	4.8%	111 ± 103	6.3%	-4.5%	108 ± 103	6.1%	60.6%	44705	0.3%	101 ± 1	7.7%	27.1%
	381	9.6%	160 ± 202	9.8%	-9.4%	129 ± 104	8.6%	77.6%	81281	0.5%	101 ± 3	8.5%	39.7%
7	401	2.5%	199 ± 571	4.3%	5.8%	113 ± 84	4.1%	53.7%	396385	0.1%	111 ± 18	3.3%	31.2%
	801	5.0%	140 ± 213	4.9%	5.6%	170 ± 273	6.0%	67.6%	726705	0.3%	113 ± 16	4.2%	41.4%
	1601	9.9%	236 ± 243	12.6%	-2.9%	239 ± 347	10.7%	81.7%	1321281	0.5%	114 ± 19	4.1%	55.0%
8	1626	2.5%	196 ± 285	5.8%	15.4%	137 ± 205	4.3%	59.8%	6392215	0.1%	120 ± 19	2.2%	41.0%
	3251	5.0%	421 ± 425	12.6%	10.3%	507 ± 664	14.3%	70.1%	11719060	0.3%	113 ± 13	2.5%	52.4%
	6501	10.0%	$753 \pm 1,916$	17.2%	7.4%	$858 \pm 1,425$	20.7%	80.7%	21307381	0.5%	114 ± 15	2.5%	63.1%

that more gates have to be employed to obtain an approximate circuit which minimizes two error metrics simultaneously.

C. Parameters of the evolved multipliers

The parameters of the best evolved approximate multipliers are shown in Figure 2. Firstly, we will compare the methods in terms of the achieved error and average error magnitudes (second row). Contrasted with S2, S1 produces multipliers showing one order of magnitude better average error for the same worst case error. For example, $e_{avg\%} = 0.42$ for S1 and $e_{avg\%} = 3.0$ for S2 ($w = 8$ and $e_{wst\%} = 10$). However, S3 is capable of discovering solutions that have even better parameters. For example, $e_{wst\%} = 2.2$ for S3, but $e_{wst\%} = 10$ for S1 ($w = 8$ and $e_{avg\%} = 0.43$). If we compare the solutions obtained by S2 and S3 on the same interval (i.e. $e_{avg\%} < 0.5$) the values of error magnitude are very similar ($e_{wst\%} < 2.0$).

The first row of Figure 2 depicts the achieved reduction in the approximate circuit's size. From this perspective, strategy S1 exhibits the worst results. The ratio of reduced gates is noticeably lower when compared to the other two scenarios. A significant reduction of the number of utilized gates can be seen for S2 and S3. The obtained reduction increases with increasing of the error rate, but its maximal value decreases with decreasing the bit-with. This fact is probably caused by decreasing the number of available gates.

The improvement in power consumption is shown in the third row of Figure 2. We can identify that power consumption decreases with the increasing number of removed gates regardless of the utilized method. However, this dependency is observable only for such cases in which the number of reduced gates exceeds approx. 20%. Below this level, the approximate multipliers exhibit the same or even worst power consumption compared to the original fully functional multiplier. This behavior is noticeable especially in the case of the multipliers produced by S1 scenario. There are 4-bit multipliers that require even more than 1.5 higher amount of power.

To conclude this evaluation, it is evident that S1 represents a scenario which does not offer any advantage. The evolved solutions do not improve power consumption and even if S1 produces the results with one order of magnitude better average error than S2, the similar values can be obtained using S3.

The obtained results are promising when a comparison with the results from literature is made. For example, 4-bit (8-bit) multiplier with $e_{avg\%} = 2.6$, $e_{wst\%} = 22.2$, $pwr_{impr\%} = 13$ ($e_{avg\%} = 3.25$, $e_{wst\%} = 22.2$, $pwr_{impr\%} = 33.1$) was obtained by combining several manually created 2-bit approximate multipliers [9]. Using our approach, one order of magnitude better errors were achieved for the equivalent power reduction (see results of S3). The parameters of the best evolved 4-bit (8-bit) multiplier are $e_{avg\%} = 0.41$, $e_{wst\%} = 5.33$ for $pwr_{impr\%} = 11.38$ ($e_{avg\%} = 0.12$, $e_{wst\%} = 0.63$ for $pwr_{impr\%} = 31.6$).

The authors of SALSA reported 80% reduction in power consumption for 8-bit multiplier with $e_{wst\%} = 10$ [6]. In our case, 96% power reduction was obtained by scenario S2 for the same error level. Using SASIMI approach, approx. 45% power improvement was achieved for 8-bit multiplier with $e_{avg\%} = 0.5$ [7]. A solution with the same average error magnitude showing 79% power reduction was discovered by S3. However, as the authors utilized a different technology and unspecified fully functional multipliers, it is not feasible to provide a fair comparison of the achieved power reduction.

V. CONCLUSIONS

In this paper, a two-phase design method based on CGP optimized for synthesis of approximate circuits was introduced and evaluated. We provided an analysis of three error-oriented design scenarios and discussed their properties from the perspective of the obtained results. In addition to that, it was shown that the proposed design technique is able to generate

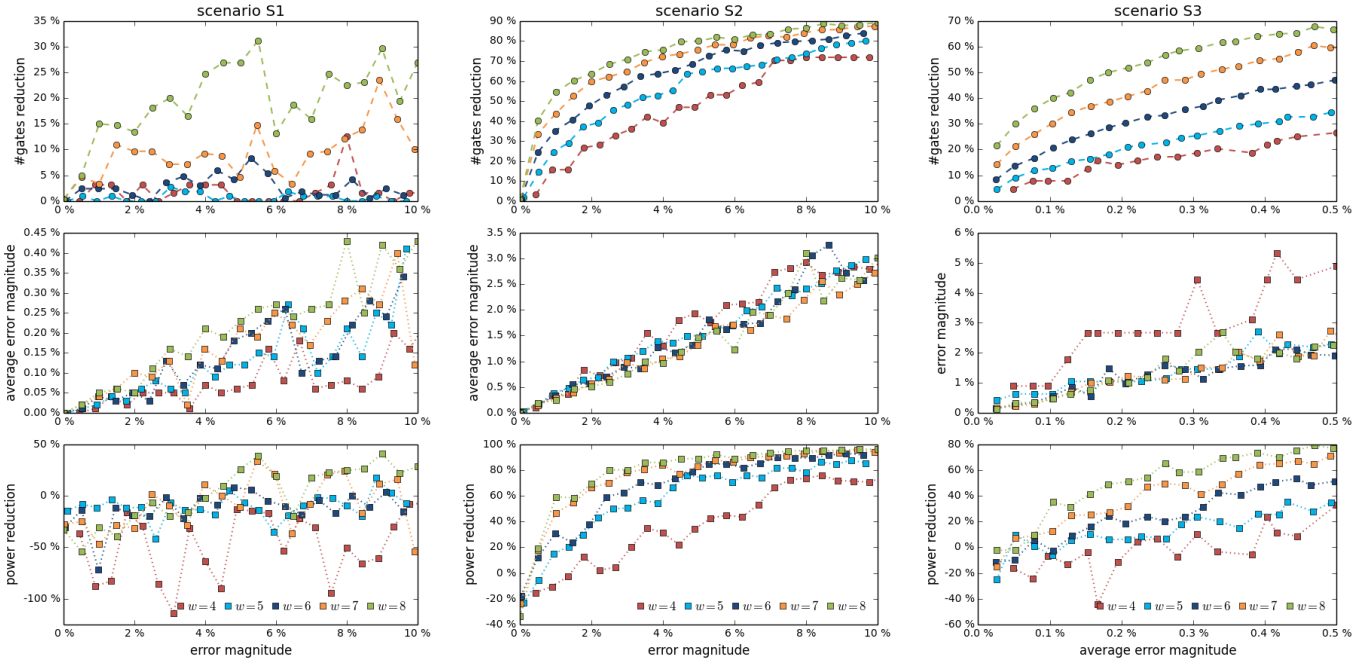

Fig. 2. Parameters of the best evolved approximate multipliers produced by the proposed optimization scenarios

multipliers that are unreachable using the technique based on combining of small 2-bit approximate multipliers.

Our future research will be focused on a detailed comparison of various approaches to the approximate circuit design and reducing of the computation requirements of the evolutionary approach.

VI. Acknowledgments

This work was supported by the Czech science foundation project 14-04197S, Brno University of Technology project FIT-S-14-2297 and the IT4Innovations Centre of Excellence CZ.1.05/1.1.00/02.0070.

References

[1] V. Venkatachalam and M. Franz, "Power reduction techniques for microprocessor systems," *ACM Computing Surveys*, vol. 37, no. 3, pp. 195–237, 2005.

[2] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 8–23, 2013.

[3] D. Shin and S. K. Gupta, "Approximate logic synthesis for error tolerant applications," in *Design, Automation and Test in Europe, DATE 2010*. IEEE, 2010, pp. 957–960.

[4] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2011, pp. 667–673.

[5] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. of the 18th IEEE European Test Symposium*. IEEE, 2013, pp. 1–6.

[6] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: systematic logic synthesis of approximate circuits," in *The 49th Annual Design Automation Conference 2012, DAC '12*. ACM, 2012, pp. 796–801.

[7] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Design, Automation and Test in Europe, DATE'13*. EDA Consortium San Jose, CA, USA, 2013, pp. 1–6.

[8] L. Sekanina and Z. Vasicek, "Approximate circuits by means of evolvable hardware," in *2013 IEEE International Conference on Evolvable Systems*, ser. Proceedings of the 2013 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE CIS, 2013, pp. 21–28.

[9] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electronics*, vol. 7, no. 4, pp. 490–501, 2011.

[10] J. D. Lohn and G. S. Hornby, "Evolvable hardware: Using evolutionary computation to design and optimize hardware systems," *IEEE Computational Intelligence Magazine*, vol. 1, no. 1, pp. 19–27, 2006.

[11] P. C. Haddow and A. M. Tyrrell, "Challenges of evolvable hardware: past, present and the path to a promising future," *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 183–215, 2011.

[12] L. Sekanina, "Evolvable hardware," in *Handbook of Natural Computing*. Springer Verlag, 2012, pp. 1657–1705.

[13] J. F. Miller, "On the filtering properties of evolved gate arrays," in *1st NASA-DoD Workshop on Evolvable Hardware*. IEEE Computer Society, 1999, pp. 2–11.

[14] T. Knieper, P. Kaufmann, K. Glette, M. Platzner, and J. Torresen, "Coping with resource fluctuations: The run-time reconfigurable functional unit row classifier architecture," in *Proc. of the 9th Int. Conf. on Evolvable Systems: ¿From Biology to Hardware*, ser. LNCS, vol. 6274. Springer, 2010, pp. 250–261.

[15] J. Petrlik and L. Sekanina, "Multiobjective evolution of approximate multiple constant multipliers," in *IEEE International Symposium on Design and Diagnostics of Electronic Circuits and Systems*. IEEE Computer Society, 2013, pp. 116–119.

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

[17] E. M. Sentovich, "Sis: A system for sequential circuit synthesis, University of California, Berkeley," 1992.

[18] Z. Vasicek and K. Slany, "Efficient phenotype evaluation in cartesian genetic programming," in *Proc. of the 15th European Conference on Genetic Programming*, ser. LNCS 7244. Springer Verlag, 2012, pp. 266–278.