On Evolutionary Synthesis of Linear Transforms in FPGA

Zdeněk Vašíček, Martin Žádník, Lukáš Sekanina and Jiří Tobola

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
{vasicek, izadnik, sekanina, itobola}@fit.vutbr.cz

Abstract. In this paper, an evolutionary approach is used to design multiple constant multipliers (MCMs). As these circuits can be composed of adders, subtractors and shifters, they perform a linear transform. An important consequence is that only a single input value is sufficient to completely evaluate a candidate circuit independently of its size and the bit width of the datapath. Proposed method is able to compete with well-optimized heuristics in particular problem instances. This paper also deals with a hardware acceleration of the method in FPGA which provides the speedup of two orders of magnitude in comparison with a conventional PC.

1 Introduction

By *evolvable hardware* we mean the usage of evolutionary algorithms (EA) either for hardware design or for dynamic hardware adaptation [1]. While in the first case the goal is to automatically generate innovative solutions (i.e. the evolutionary algorithm is used only in the design phase of a product), the second case deals with the online modification of a reconfigurable structure. The objective is to improve the performance of the system working in a changing environment or repair the system when faults are present. This paper is devoted to the evolutionary design of digital circuits in which complete circuit structures are evolved. We will not deal with evolutionary circuit optimization in which only values of some parameters of a human-predesigned solution are sought by evolutionary algorithm.

1.1 Evolutionary Circuit Design

In some areas, innovative circuits were designed using EAs, see for example [2, 3, 4, 1]. In this context, the term *innovative* means that the solution exhibits a better quality in some aspects wrt existing designs of the same category. For example, the solution would occupy a smaller area on a chip, compute faster, provide a better precision, reduce energy consumption, increase the reliability etc.

However, the evolutionary circuit design is not competitive in all design problems because of the so called *scalability problems*. From the viewpoint of the

scalability of representation, the problem is that long chromosomes which are usually required to represent complex solutions imply large search spaces that are typically difficult to search. In many cases, even a well tuned evolutionary algorithm fails to find an innovative solution in a reasonable time.

Another problem is related to the fitness calculation time. In case of the combinational circuit evolution, the evaluation time of a candidate circuit grows exponentially with the increasing number of inputs (assuming that all possible input combinations are tested in the fitness function). Hence, the evaluation time becomes the main bottleneck of the evolutionary approach when complex circuits with many inputs are evolved. This problem is known as the problem of *scalability of evaluation*. In order to reduce the time of evaluation, various techniques have been adopted:

- Only a subset of all possible input vectors is utilized. That is typical for evolution of filters, classifiers or robot controllers [1]. Unfortunately, nobody can guarantee a correct circuit behavior for those input combinations which were not used during evolution. Hence, evolved circuits are validated at the end of evolution using a test set — a representative set of input vectors which differ from the training set.
- In some cases it is sufficient to evaluate only some structural properties of candidate circuits which can be done with a reasonable time complexity. For example, because the testability of a candidate circuit can be calculated in a quadratic time, very large benchmark circuits with predefined testability properties can be evolved [5].

An obvious conclusion is that the perfect evaluation procedures are applicable only for small circuits. On the other hand, when more complex circuits are evolved, only an imperfect fitness calculation method can be used. (Note that although some resulting circuits can not be considered as general solutions, they can be quite effective from the practical point of view.) This strongly contrasts with conventional methodologies which are able to solve large instances of synthesis problems and which ensure that all circuits are perfectly functional.

1.2 Linear Transforms

A question is whether one can escape, at least for some problems, from this conclusion. In order to evolve large circuits and simultaneously perform a perfect evaluation, we have to identify such design problems for which the evaluation of a candidate solution requires either constant or linear time with respect to the number of inputs/components. We can observe that if a candidate circuit consists only of linear operators (such as addition, subtraction, shift etc.) and the goal is to implement a linear transform, then the circuit can be completely evaluated using *a single test vector* independently of the number of inputs and the bit width of the datapath.

A multiple constant multiplier (MCM) is a digital circuit which multiplies its single input x by N constants and so it generates N output products. As

this circuit can be composed of adders, subtractors and shifters, it is very useful for low-power implementations of FIR filters in which the input signal has to be multiplied by different, but constant values [6]. An important characteristics of MCM design is that because the transform is linear, only a single input value of x (e.g. $x = 1$) is sufficient to completely evaluate a candidate circuit. This unique feature has not been utilized in the evolvable hardware field although there are many works dealing with the evolutionary design of multiplierless filters (in which the fitness calculation is based on sampling the frequency characteristics and calculating the difference from the required frequency characteristics [7, 8]) and constant-coefficient multipliers (in which a symbolic verification algorithm is used [3, 9]).

Finding the optimal solution of the MCM problem, i.e., the one with the fewest number of components (in particular, additions and subtractions) is known to be NP-complete. A very efficient heuristic algorithm for the MCM problem was recently published [10]. This is a graph-based algorithm which can handle problem sizes as large as one hundred 32-bit constants. The algorithm can be considered as the state of the art method for the MCM design problem.

1.3 The Aim of This Paper

The goal of this paper is to show that when a candidate solution can be evaluated in a linear time (or in a constant time when the evaluation is parallelized in hardware), very large and simultaneously precisely-evaluated circuit designs can be evolved. In particular, an evolutionary algorithm loosely inspired by Cartesian Genetic Programming (CGP) [2] is employed to generate innovative implementations of MCMs. In comparison with the heuristics [10], the proposed evolutionary-based approach is able to reduce the delay of MCMs and the number of shift operators.

In order to accelerate the evolutionary design as much as possible, a complete evolutionary system was implemented in an FPGA. As the proposed system is able to evaluate a candidate circuit in a single clock cycle, it is necessary to generate a new chromosome in a single clock cycle. An evolutionary algorithm implemented as a program running on an on-chip processor (Microblaze or PowerPC), however, is not able to achieve this frequency. Hence a new evolutionary algorithm has to be implemented as an application-specific circuit in the FPGA. This implementation provides a significant speedup of the evolutionary MCM design in contrast to a conventional PC.

The paper is organized as follows. Proposed evolutionary approach is introduced in Section 2. Section 3 describes the FPGA solution used to accelerate the evolutionary design process. Results of experiments are presented in Section 4. Conclusions are given in Section 5.

2 Proposed Method

The goal is to synthesize a multiple constant multiplier which generates N output values $c_1 x \ldots c_N x$, where $c_1 \ldots c_N$ are given constants and x is the only input

variable. The circuit is composed of linear components – adders, subtractors and shifters. In addition to a perfect functionality, the number of components is optimized. The problem is approached using evolutionary algorithm in which the problem representation is borrowed from the CGP.

In CGP, a candidate circuit is represented as an array of u (columns) $\times$ v (rows) of programmable elements (nodes). The number of inputs, n_i, and outputs, n_o, is fixed. For MCMs, $n_i = 1$ and $n_o = N$. Feedback is not allowed. Each node input can be connected to the output of a node placed in the previous L columns or to some of program 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. Each node is programmed to perform one of functions defined in the set Γ. For MCMs, Γ includes the addition, subtraction, shifts and identity function. These functions as well as all connections are used over b bits, where $b = 16$ in our case. As Figure 1 shows, while the size of chromosome is fixed, the size of phenotype is variable (i.e. some nodes are not used). Every individual is encoded using $u \times v \times 3 + n_o$ integers.

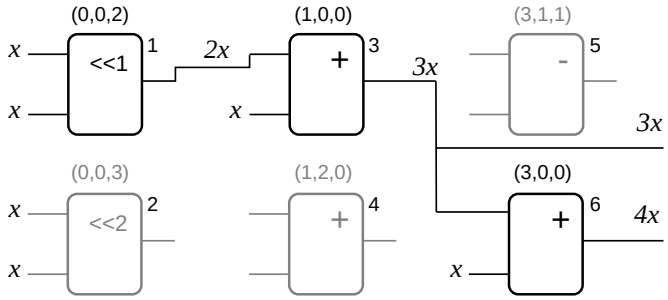

Fig. 1. Example of a candidate circuit. CGP parameters are as follows: $L = 3$, $u = 3$, $v = 2$, $\Gamma = \{$add (0), sub (1), 1b-shift (2), 2b-shift (3)$\}$. Nodes 2, 4 and 5 are not utilized. Chromosome: 0,0,2, 0,0,3, 1,0,0, 1,2,0, 3,1,1, 3,0,0, 3, 6. The last two integers indicate the outputs of the MCM. The input x is encoded as 0.

EA operates with the population of λ individuals. The initial population is randomly generated. Every new population is generated using tournament selection and mutation. In the fitness calculation, the goal is to minimize the difference between circuit outputs and required products. When a functionally perfect solution is obtained, the number of components is optimized. The evolution is stopped when the best fitness value stagnates or the maximum number of generations is exhausted.

3 FPGA Acceleration

The accelerator consists of a genetic unit, array of reconfigurable elements and fitness calculation unit. Instead of a direct reconfiguration of the FPGA, so-called Virtual Reconfigurable Circuit (VRC) is reconfigured. VRC is an application-specific reconfigurable circuit implemented on the top of the FPGA according to scheme proposed in [11]. The VRC, in fact, implements the representation used in CGP. The main execution loop (selection, genetic operations and fitness evaluation) is carried out in the FPGA while the initialization and user interface are controlled from the PC. The PC allows to fully control the evolutionary process in FPGA even during the execution. For example, it is possible to determine and set a new probability of mutation on-the-fly.

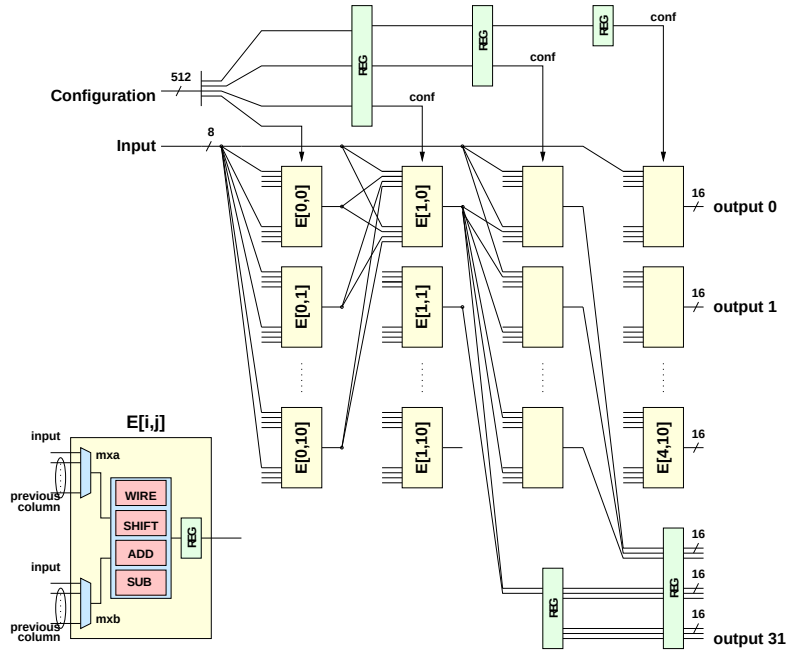

Fig. 2. VRC developed for the multiple constant multiplier design problem

3.1 Evaluation of Candidate Circuits

A candidate circuit is represented using a bitstream which defines the configuration of VRC. Figure 2 shows an example of VRC which consists of Configurable Logic Blocks (CFBs), denoted as E, placed in a grid of 4 columns and 11 rows. The inputs of each CFB can be connected either to the input value x or to the output of a CFB, which is placed anywhere in the preceding column. Each CFB

can be configured to implement one of the functions: addition, subtraction, binary shifts or identity. Each function accepts 16-bit operands and produces a 16-bit result.

In contrast with CGP, the outputs are not configurable. The number of outputs is equal to the number of CFBs. An additional mask signal is provided to determine utilized outputs. The reconfiguration is performed column by column. The computation is pipelined; a column of CFBs represents a stage of the pipeline. The configuration bitstream is stored in a register array which can hold up to 512 bits.

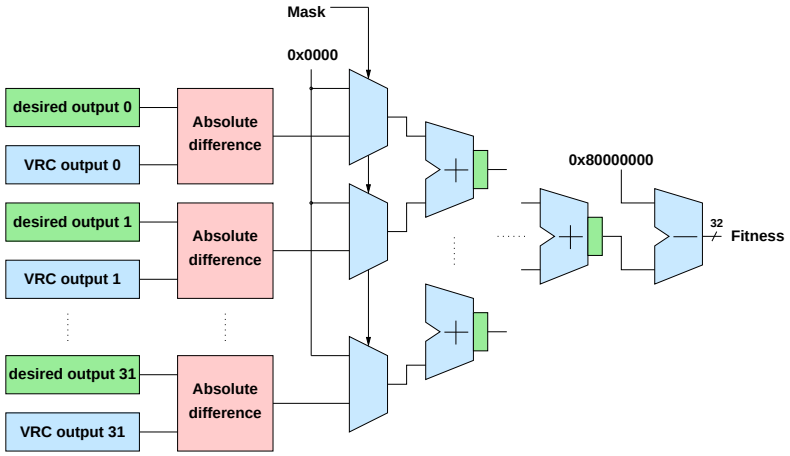

Fig. 3. Fitness calculation circuit

Figure 3 shows a circuit which is used to calculate the fitness value. As this circuit can be also pipelined, it naturally extends the pipeline existing in the VRC. The fitness value can be obtained in one clock cycle.

In order to optimize the cost of a candidate circuit (i.e. the number of used CFBs), a simple method was used. This method utilizes the fact that each CFB can implement a single wire whose implementation cost is zero. As soon as a functionally-perfect solution is found, the objective of evolutionary algorithm is to maximize the number of CFBs which operate as wires. The configuration of a single column of VRC is analyzed using comparators which return 1 in the case that a particular CFB is configured as a wire. The results of comparators are added up using a tree of adders. The sum represents the size of phenotype which is utilized as a part of the fitness value. This calculation is performed when the column of CFBs is configured.

3.2 Genetic Engine

In order to feed the VRC and fitness unit with new candidate circuits optimally, it is necessary to create a genetic unit as a specific circuit in the FPGA. Although various implementations of EAs were designed for FPGAs (e.g. [12]), we propose a new genetic unit which is able to generate a new candidate configuration every clock cycle.

Analyzing the execution loop of EA, it can be derived that there are no dependencies among chromosomes in the population. There is a backward dependency among two success generations resulting from *fitness evaluation* to *individual selection*. The hardware implementation can exploit this to highly pipeline the execution path. Moreover, operations such as mutation and crossover are very simple and lead to a fast combinatorial logic. The main contribution to the length of the pipeline is usually caused by the fitness evaluation. The overhead caused by the multiple pipeline stages is eliminated by careful setting of the population size so all stages of pipeline are fully utilized.

The FPGA design consists of three units connected in a loop: Selecting the individual, mutation/crossover and fitness evaluation. Pseudo-random number generators (PRNG) implemented as uncorrelated LFSR are used for all stochastic operations.

All the units have generically modifiable width of interface so the chromosome width can differ depending on the problem solved. It is supposed that each unit is able to accept one chromosome at the input and emit the result at the output in one clock cycle. All the units are data driven so whenever they have the input ready they perform their tasks.

Every clock cycle one individual is selected using a tournament selection and sent to a mutation unit. At the end the fitness unit evaluates a candidate chromosome and transfers it together with its fitness value to the selection unit. Following paragraphs describe the units in detail.

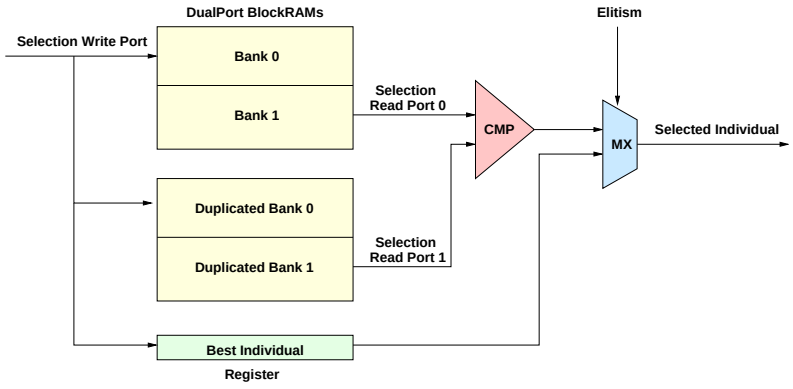

Fig. 4. Structure of selection unit

Selection unit Two randomly selected individuals are compared by their fitness value and the one with higher fitness proceeds to the next generation. Parental and offspring populations are stored in configurable BRAM memory; however, in separate banks. While the selection of parents takes place it the first bank, new individuals are already written in another bank. After the new generation is complete, the banks are switched and the evolution can continue. To fully optimize the selection process, the memory is duplicated so in one clock cycle two individuals can be accessed for the tournament. Figure 4 shows the selection unit.

Mutation Unit A generic mutation unit was designed to perform mutation according to user settings. The probability of mutation per bit, p_{bit}, is controlled by the software via writing value R into the probability register according to Equation 1.

$$R = \frac{1}{|CH| min(p_{bit}, \frac{N}{|CH|})} \cdot PRNG_{MAX} \tag{1}$$

where $|CH|$ is the length of chromosome, N is the number of samplers and $PRNG_{MAX}$ is the range of pseudo random generator (PRNG), in our case 2^{32}. Each sampler contains two pseudo-random number generators. The result of the first generator is compared with a value in the register R. If the value is higher then the mutation takes place. The second generator determines the position of mutated bit. The mutation is performed using a XOR operation over a chromosome and a mutation mask. The mutation mask is initialized to zeros and only bits where the mutation takes place are set to one.

3.3 Results of Synthesis

The evolutionary platform was described in VHDL, simulated using ModelSIM and synthesized using XILINX ISE. The architecture is parameterized in terms of the data width and the size of VRC. Results of synthesis are summarized in Table 1. As the maximum size of the chromosome is 512 bits in current implementation, the largest VRC that can be encoded consists approximately of 11×4 elements. The system was tested at 100 MHz in FPGA Virtex II Pro 2VP50ff1517. The platform is able to generate and evaluate 100 million candidate solutions per second.

4 Results

In order to evaluate proposed method, we have chosen to evolve MCMs with 3, 5, 10 and 20 16-bit constant coefficients (given in Table 2). All experiments were repeated 200 times with the population of eight individuals and five genes mutated in the chromosome. Table 2 gives other parameters of the experiments, average results (the number of generations and used adders/subtractors), the success rate and parameters of the best evolved solutions.

Table 1. Results of synthesis in Virtex II Pro 2VP50ff1517 FPGA for various VRC sizes, each of them with up to 32 outputs

	VRC 6×6			VRC 11×4		
Components	Used	Total	[%]	Used	Total	[%]
Function Generators	16085	47232	34.06	18013	47232	38.14
CLB Slices	8043	23616	34.06	9007	23616	38.14
Dffs or Latches	10280	49788	20.65	12529	49788	25.16
Block RAMs	32	232	13.79	32	232	13.79

Results are compared with the best known heuristic approach [10] which produces very compact solutions. Table 2 shows that the proposed evolutionary-based approach is able to generate multipliers that are competitive with results obtained using the state of the art heuristic approach. The evolution can reduce the total number of components as well as the delay of the designed MCMs.

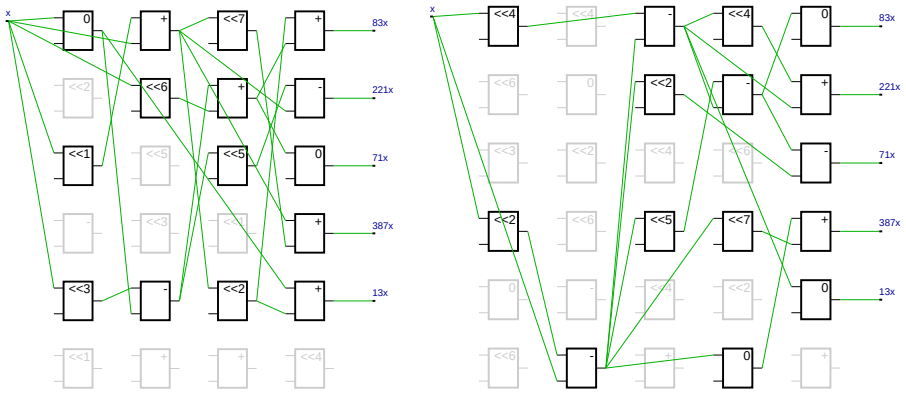

Fig. 5. Example of multipliers evolved by proposed hardware accelerator. The objective is to design multipliers which multiply the input by five coefficients (83, 221, 71, 387, 13). As the VRC utilizes 4×6 (5×6) elements, the delay of the evolved multiplier is equal to 4 (5).

Figure 5 shows two examples of evolved solutions for the 5 constant MCM and different VRCs. As it can be seen, the number of VRC columns determines the delay of the evolved multiplier. Figure 6 compares the best evolved solution with the solution provided by the heuristics for the 3 constant MCM. Evolved solution contains 2 shifters less and exhibits shorter delay than the solution provided by the heuristics. Figure 7 shows one of evolved innovative solutions for the 20 constant MCM. This circuit consists of 4 shifters and 19 adders/subtractors. The heuristic approach [10] provides a solution consisting of 19 shifters and 19 adders/subtractors. Delay remains unchanged.

Table 2. Results of evolutionary design of MCMs with different coefficients. Population size is 8. Averages are calculated from 200 independent runs.

Settings		Average Results			The Best MCM			
cols × rows	maxgen	geners.	#add/sub	succ. rate	delay	add/sub	shifts	operations
3 constants: 2925, 23111, 13781								
Heuristics [10]					8	8	8	16
5×6	20M	1M62	14	68.5	5	9	8	17
6×6	20M	1M27	14	86.5	6	8	8	16
7×4	40M	2M15	13	99.0	7	8	6	14 (Fig. 6)
5 constants: 83, 221, 71, 387, 13								
Heuristics [10]					5	6	6	12
4×6	20M	461k	10	99.5	4	7	6	13
5×6	20M	207k	11	99.5	5	6	6	12
6×6	20M	114k	11 (Fig. 5)	100.0	6	6	5	11
10 constants: 117, 1123, 743, 221, 1069, 7605, 987, 16689, 3033, 29								
Heuristics [10]					8	14	13	27
10×4	40M	4M8	23	99.0	7	15	12	27
7×6	20M	4M7	23	95.5	6	17	11	28
9×4	40M	9M5	22	91.0	9	17	9	26
20 constants: 1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71								
Heuristics [10]					4	19	8	27
4×10	40M	457k	23	100	4	19	4	23 (Fig. 7)
5×10	40M	347k	23	100	4	19	4	23
6×5	40M	772k	21	100	5	19	3	22

As producing 10M generations (the population size is 8) requires 230 sec. on a Celeron 2.4 GHz processor, the speedup provided by the accelerator is approximately 287 times. Although current implementation of the accelerator is able to design only such multipliers which fit into VRC with the configuration size up to 512 bits, we proved using a SW implementation that proposed evolutionary approach is able to produce competitive MCMs approximately up to 100 outputs.

5 Conclusions

A very time-consuming evaluation of candidate configurations is one of problems which influences the applicability of evolutionary circuit design. In this paper, we focused on such problems in which a candidate solution can be perfectly evaluated in a very short time. Linear transforms in general, and multiple constant multiplications in particular, belong to this class. Although well-optimized heuristics exist for linear transforms design, we confirmed that novel implementations of multiple constant multipliers can be designed using evolutionary algorithm. In our future work, we will further explore the potential of proposed accelerator, especially in searching for larger MCMs optimized for area as well as delay.

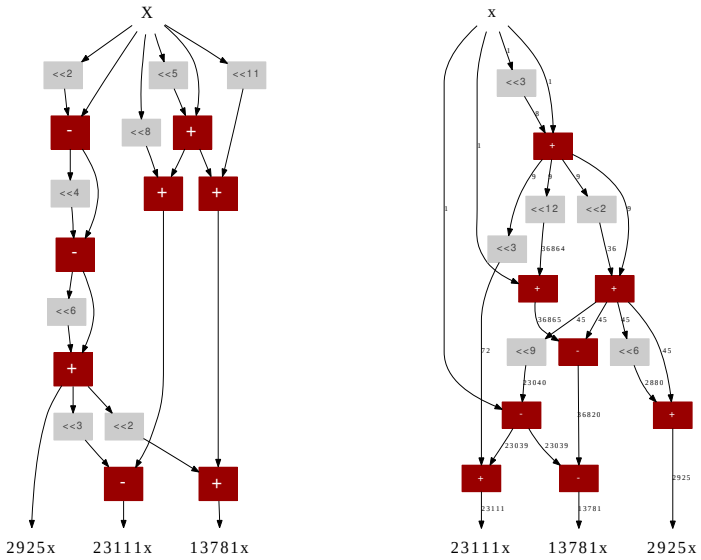

Fig. 6. MCM with 3 coefficients (2925, 23111, 13781): according to [10] (left), the best evolved solution (right)

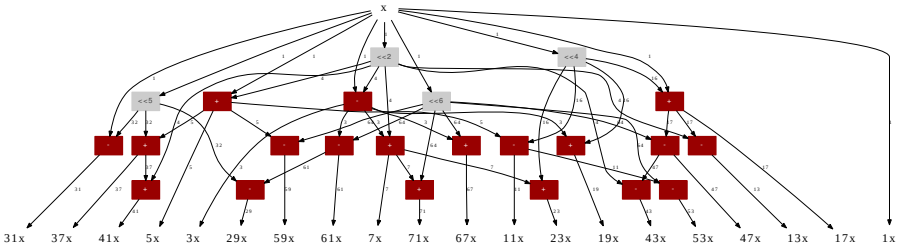

Fig. 7. The best evolved MCM with 20 constant coefficients

Acknowledgements

This research was partially supported by the Grant Agency of the Czech Republic under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* and the Research Plan No. MSM 0021630528 – *Security-Oriented Research in Information Technology.*

References

[1] Higuchi, T., Liu, Y., Yao, X.: Evolvable Hardware. Springer (2006)
[2] Miller, J., Job, D., Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines **1**(1) (2000) 8–35
[3] Aoki, T., Homma, N., Higuchi, T.: Evolutionary Synthesis of Arithmetic Circuit Structures. Artificial Intelligence Review **20**(3–4) (2003) 199–232
[4] Higuchi, T., Iwata, M., Keymeulen, D., Sakanashi, H., Murakawa, M., Kajitani, I., Takahashi, E., Toda, K., Salami, M., Kajihara, N., Otsu., N.: Real-World Applications of Analog and Digital Evolvable Hardware. IEEE Transactions on Evolutionary Computation **3**(3) (1999) 220–235
[5] Pecenka, T., Kotasek, Z., Sekanina, L., Strnadel, J.: Automatic discovery of RTL benchmark circuits with predefined testability properties. In: 2005 NASA / DoD Conference on Evolvable Hardware, IEEE Computer Society (2005) 51–58
[6] Meyer-Baese, U.: Digital Signal Processing with Field Programmable Gate Arrays. Springer (2004)
[7] Erba, M., Rossi, R., Liberali, V., Tettamanzi, A.: An evolutionary approach to automatic generation of VHDL code for low-power digital filters. In: Genetic Programming, Proceedings of EuroGP'2001. Volume 2038., Springer-Verlag (2001) 36–50
[8] Hounsell, B.I., Arslan, T., Thomson, R.: Evolutionary design and adaptation of high performance digital filters within an embedded reconfigurable fault tolerant hardware platform. Soft Computing **8**(5) (2004) 307–317
[9] Homma, N., Aoki, T., Higuchi, T.: Multiplier block synthesis using evolutionary graph generation. In: 6th NASA / DoD Workshop on Evolvable Hardware (EH 2004), IEEE Computer Society (2004) 79–82
[10] Voronenko, Y., Puschel, M.: Multiplierless multiple constant multiplication. ACM Transactions on Algorithms **3**(2) (2007) 1–282
[11] Sekanina, L.: Virtual reconfigurable circuits for real-world applications of evolvable hardware. In: Evolvable Systems: From Biology to Hardware, Fifth International Conference, ICES 2003. Volume 2606 of LNCS., Trondheim, Norway, Springer-Verlag (2003) 186–197
[12] Shackleford, B.: A high-performance, pipelined, FPGA-based genetic algorithm machine. Genetic Programming and Evolvable Machines **2**(1) (2001) 33–60