Analysis of Reconfigurable Logic Blocks for Evolvable Digital Architectures

Lukas Sekanina and Petr Mikusek

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz, imikusek@fit.vutbr.cz

Abstract. In this paper we propose three small instances of a reconfigurable circuit and analyze their properties using the brute force method and evolutionary algorithm. Although proposed circuits are very similar, significant differences were demonstrated, namely in the number of unique designs they can implement, the sensitiveness of functions to the inversions in the configuration bitstream and the average number of generations needed to find a target function. These findings are quite unintuitive. Once important (sensitive) bits of the reconfigurable circuit are identified, evolutionary algorithm can incorporate this knowledge. We believe that the proposed type of analysis can help those designers who develop new reconfigurable circuits for evolvable hardware applications.

1 Introduction

One of possible approaches to building adaptive hardware is to combine reconfigurable hardware with search algorithms. In the field of evolvable hardware, the evolutionary algorithm is used to find a suitable configuration of a reconfigurable device [1, 2].

In the area of digital circuits, application-specific reconfigurable circuits and field programmable gate arrays (FPGA) can be considered as the most popular reconfigurable platforms for evolvable hardware. In general, the reconfigurable digital circuit consists of an array of programmable logic elements, programmable interconnects and programmable I/O ports. The function of programmable logic elements and their interconnection (i.e. the circuit functionality) is defined using a configuration bitstream. The configuration bitstream is stored in a configuration register (or memory) whose bits directly control the configurable switches and multiplexers of the platform.

When the evolvable system is completely implemented on a single chip, a part of the chip is devoted for evolving designs and another part is used to implement the evolutionary algorithm. In these systems, evolutionary algorithm usually directly operates with the configuration register, i.e. the chromosome is considered as a candidate configuration. A kind of internal reconfiguration has to be employed (for example, ICAP in Xilinx Virtex II+ families [3]). Another option is to configure the reconfigurable device externally, for example, from a PC where the evolutionary algorithm is implemented [4]. The quality of evolved solutions

M. Giacobini et al. (Eds.): EvoWorkshops 2008, LNCS 4974, pp. 144–153, 2008.

depends on the evolutionary algorithm as well as the reconfigurable device. As this paper primarily deals with reconfigurable hardware in evolvable digital architectures, we will focus our attention only on the reconfigurable circuit.

When one is building a new reconfigurable ASIC, the reconfigurable circuit can be designed exactly according to requirements of a given application. Designer can choose the optimal type and count of configurable logic elements, suitable interconnecting network as well as configuration subsystem (organization of the configuration memory, the style of reconfiguration etc.).

When one is building a reconfigurable device with the FPGA, there are two options. (1) Evolution can work at the level of logic blocks available in the FPGA. In other words, it operates directly with the configuration bitstream of the FPGA [4, 3]. This solution requires the knowledge of the internal structure of the FPGA and the configuration bitstream. It is usually very efficient in terms of resources; however, it can be slow. (2) A new reconfigurable circuit is created on the top of an FPGA [5]. Using this method, sometimes called Virtual Reconfigurable Circuit (VRC), a very efficient reconfigurable device can be created for a given application. However, its implementation cost can be significant, as everything must be implemented using resources available in the FPGA.

In both cases, designer has to come up with a suitable configurable logic blocks and configurable interconnections with respect to the target application. Designer has to define the organization of the configuration register (memory) in order to maximize the efficiency of evolutionary algorithm.

The goal of this paper is to demonstrate how these design choices can influence the class of functions which will be implementable in the particular reconfigurable circuit and the efficiency of a search algorithm in the space of possible configurations. In particular, we will be interested in those architectures in which the reconfiguration subsystem is implemented using multiplexers, i.e. the function of a configurable element as well as the interconnection is determined using multiplexers. This is typical for reconfigurable ASICs as well as for VRCs. In this study we will consider a simple reconfigurable circuit with four inputs and two outputs and a 32-bit configuration register. As the number of possible configurations is relatively small (2^{32}), we can analyze its behavior by brute force to see how its structure influences the number of implementable designs, the efficiency of the search algorithm and the effect of mutations in the configuration bitstream. Results of the analysis can be exploited for designing new evolvable systems for real world-applications in which the small reconfigurable circuit can represent a single reconfigurable logic block of a complex reconfigurable system.

2 Proposed Model of a Reconfigurable Circuit

In order to perform the analysis of a typical reconfigurable unit observable in current evolvable hardware systems, we propose to investigate the structure and properties of a small instance of VRC. The VRC is used to implement a combinational logic circuit of four inputs (x_0, x_1, x_2, x_3) and two outputs (y_0, y_1) whose

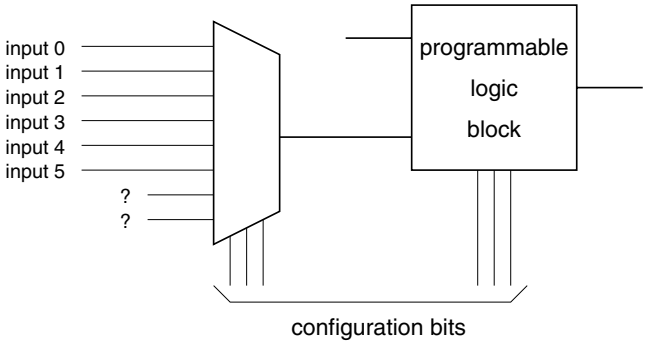


Fig. 1. A non-optimized use of multiplexers

function is defined using 32 bits. The VRC contains four logic blocks. Obviously, such circuit can be implemented using two 16-bit look-up tables. However, that type of implementation would not allow us to estimate the behavior of larger instances of VRCs which exhibit many similar features with the proposed architecture.

Because it is supposed that the chromosome directly represents the configuration bits of the VRC, all possible bit combinations should represent valid circuits. Moreover, in order to make the evolution efficient, the implementation of EA easier and the utilizations of hardware resources economical, it is desirable to perfectly utilize all possible combinations of groups of bits. Consider the example given in Figure 1. The 8-input multiplexer effectively uses only six inputs; the last two inputs has to be connected somewhere. Anyway, three bits must be included in the chromosome to control the multiplexer's selector. Then, the two out of eight combinations are not used effectively, which can turn the search algorithm to a wrong part of the search space.

For comparisons, we propose three architectures of VRC, labeled as cfg4f, cfg8f and cfg16f. They have the same number of inputs, outputs, configurable blocks and the size of configuration register. They differ in the number of functions supported by configurable blocks and the reconfiguration options.

2.1 Reconfigurable Circuit cfg4f

Figure 2 shows the reconfigurable circuit cfg4f which consists of four programmable elements B0–B3 and three stages of multiplexers. Each of configurable blocks can implement four different functions. The first stage of multiplexers selects a primary input which will be connected to blocks B0 and B1. As there are only six possible input points for the second stage of multiplexers (four primary inputs and the outputs of B0 and B1), the 8-input multiplexers can not be utilized perfectly. Hence, the output of block B0 and B1 is connected to the multiplexers twice. From the point of evolutionary design, the probability that a connection is made between block B0/B1 and the second stage of multiplexers is higher than for the primary inputs and the second stage of multiplexers. The

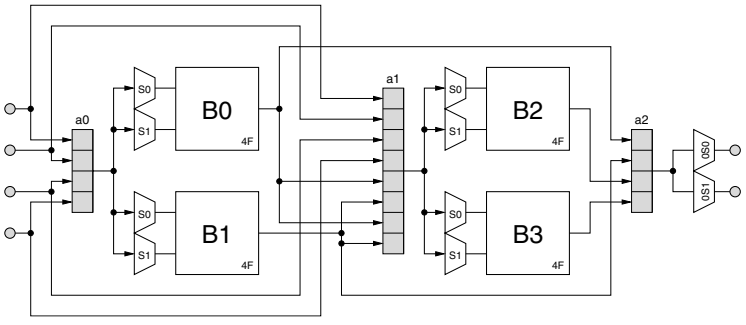


Fig. 2. Reconfigurable circuit cfg4f

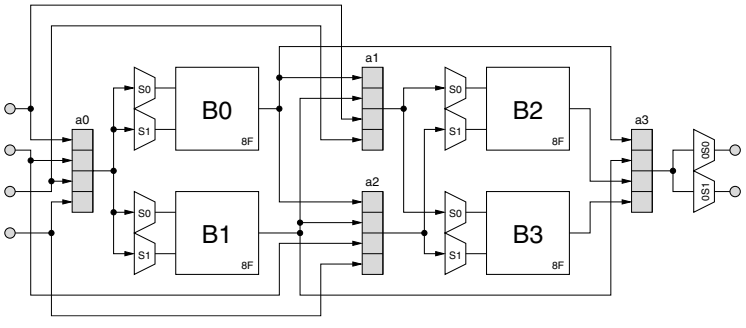


Fig. 3. Reconfigurable circuit cfg8f

primary outputs can be connected either to blocks B0, B1, B2 or B3 using the third stage of multiplexers. Selection bits of the third stage of multiplexers are perfectly utilized. For next comparisons, we will consider three variants of cfg4f which differ in the function sets supported in configurable blocks:

- cfg4f(xornot) utilizes functions NAND(0), NOR(1), XOR(2) and NOT(3)
- cfg4f(xoror) utilizes functions NAND(0), NOR(1), XOR(2) and OR(3)
- cfg4f(xorxnor) utilizes functions NAND(0), NOR(1), XOR(2) and XNOR(3)

2.2 Reconfigurable Circuit cfg8f

Figure 3 shows reconfigurable circuit cfg8f which employs configurable blocks with eight functions (NOR (0), x AND $\bar{y}$ (1), $\bar{x}$ AND y (2), AND (3), OR (4), $\bar{x}$ OR y (5), x OR $\bar{y}$ (6), NAND (7)). Because three bits of the configuration bitstream are devoted to the selection of a function, fewer bits can be used to define the interconnects.

The first and third stage of multiplexers is identical with cfg4f. As the second stage uses 4-input multiplexers, not all six possible points (primary inputs and the outputs of blocks B0 and B1) can be connected to block B2 and B3. Hence,

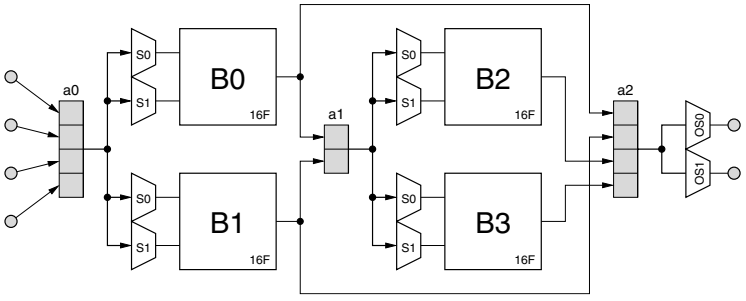


Fig. 4. Reconfigurable circuit cfg16f

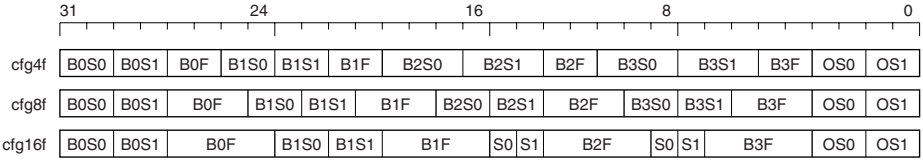


Fig. 5. Structure of the configuration bitstreams. Notation: B_x – the configurable block, S_x – the selector, F – function definition, O – the primary output.

block B2 can read its inputs from the primary input x_0 and x_2 or from blocks B0 and B1. Block B3 can read its inputs from the primary input x_1 and x_3 or from blocks B0 and B1.

2.3 Reconfigurable Circuit cfg16f

Similarly to cfg8f, also cfg16f restricts the interconnection options. Blocks B2 and B3 can be connected only with blocks B0 or B1. On the other hand, this allows the use of a full repertoire of possible logic functions over two logic variables in all configurable blocks. Figure 4 shows architecture of cfg16f. Finally, the structure of configuration bitstreams of all circuits is given in Figure 5.

3 Experimental Evaluation

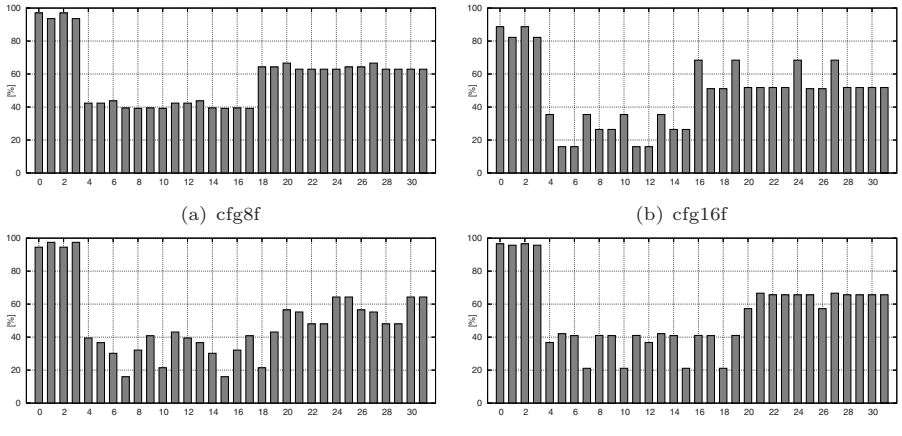
In order to analyze the behavior of proposed reconfigurable circuits a well-optimized software simulator was created. As a single configuration can be evaluated in approx. 100 ns, it is possible to test all configurations in less than 8 minutes on a common PC. The number of different configurations is $|C| = 2^{32}$. The theoretical number of possible logic behaviors is $|F| = 2^{n_o \cdot 2^{n_i}}$, where n_i is the number of primary inputs and n_o is the number of primary outputs, i.e. $2^{2.2^4} = 2^{32}$ in our case. However, the number of logic functions which can be implemented in the reconfigurable circuit is much lower because two and more different configurations quite often represent the same logic behavior. This is typical for all reconfigurable devices used for evolvable hardware.

Table 1. Characterization of reconfigurable circuits in terms of the number of different (unique) logic functions which can be implemented and the number of different implementations of a single logic function (occurrence)

Circuit	cfg4f (xornot)	cfg4f (xoror)	cfg4f (xorxnor)	cfg8f	cfg16f
Unique designs	57,837	119,502	104,468	178,764	57,712
Avr. occurence	74,260	35,941	41,113	24,026	74,421
Max. occurence	86,994,432	86,926,336	95,109,120	113,224,704	308,514,048
Min. occurence	256	128	256	64	256
Designs with min. occ.	13,272	1,512	29,160	29,952	15,864

3.1 Achievable Functions

First series of experiments is devoted to characterizing proposed reconfigurable circuits in terms of the number of unique designs (logic functions) and the occurrence of some specific designs. Table 1 shows that the number of unique designs (in the space of 2^{32} possible designs) is quite small. Circuit cfg8f provides the highest number of unique designs (178,764). We can observe how significantly the number of unique designs decreases when only one of functions in configurable blocks is changed from the two-input OR to the single input NOT. Some functions are very frequent on all reconfigurable circuits, for example $y_0 = y_1 = 0$, $y_0 = y_1 = 1$, or $y_0 = y_1 = x_k$. No function exists which can be implemented uniquely; the minimum number of occurrences of a function is 64. There are only 8,888 different functions which can be implemented on all five variants of the circuit.



(a) cfg8f (b) cfg16f

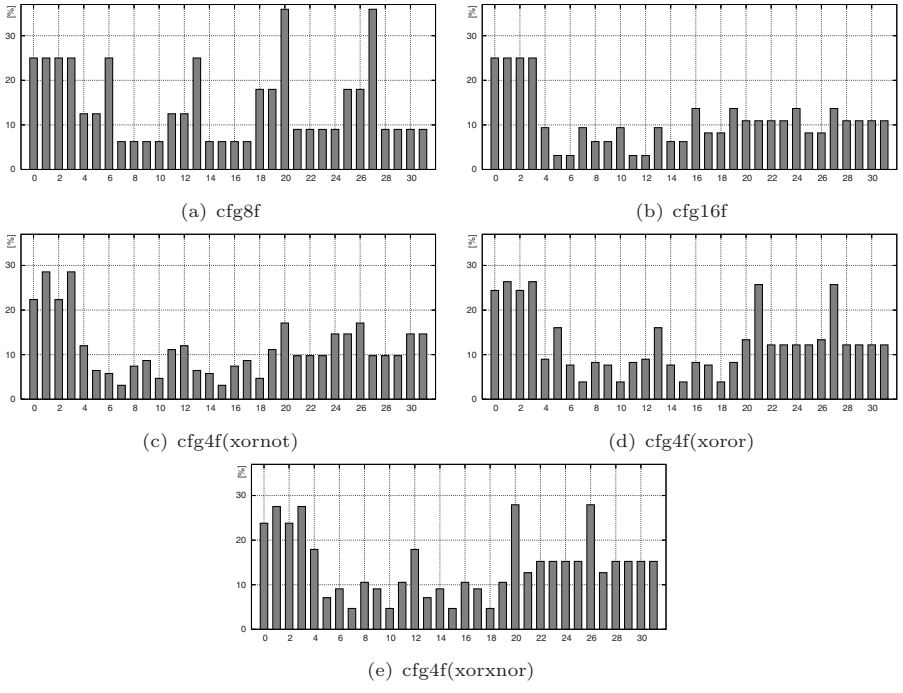
(c) cfg4f(xornot) (d) cfg4f(xoror)

Fig. 6. The number of changes in logic functions when jth bit is inverted (calculated for all configurations)

3.2 The Sensitivity of Functions to Inversions

As mutation is usually implemented using the inversion of a particular bit, it is important to analyze to what extent the circuit function is sensitive to bit inversions of the configuration bitstream. For every configuration c_i, we calculated the corresponding logic function $f_i(c_i)$. Then, for every single independent inversion of the configuration bit $j, j \in \{1, \ldots, 32\}$, we checked whether the new function $f_i(c_i)^{(j)}$ is different from $f_i(c_i)$. Figure 6 shows how many times (in percentage points) the logic function is changed when jth bit of the configuration c_i is inverted ($i \in \{1, \ldots, 2^{32}\}$). A general observation is that independently of the reconfigurable circuit and its configuration, we can see that the logic function is changed in more than 90 % cases when bit 0, 1, 2 or 3 are inverted. The circuit function is also very sensitive to other four bits in cfg16f. Other bits do not seem to be so important. Results are not shown for cfg4f(xorxnor) because they are indistinguishable from cfg4f(xoror) in Figure 6.

Figure 7 shows the results of the same experiment; however, the y-axis does not give the number of changes in logic functions. It displays the sum of Hamming distances (in percentage points, the maximum is $2^{32}.2^{n_o}.2^{n_i}$) between truth tables of original logic functions and truth tables of logic functions obtained using the inversion of jth bit. Thus, we can see how significant the inversions are for



(a) cfg8f

(b) cfg16f

(c) cfg4f(xornot)

(d) cfg4f(xoror)

(e) cfg4f(xorxnor)

Fig. 7. The sum of Hamming distances (in percentage points, the maximum is $2^{32}.2^{n_o}.2^{n_i}$) between truth tables of original logic functions and truth tables of logic functions obtained using the inversion of jth bit

Table 2. The average percentages of changes in logic functions and Hamming distances of truth tables over all configurations and independent bit inversions

Circuit	cfg4f (xornot)	cfg4f (xoror)	cfg4f (xorxnor)	cfg8f	cfg16f
Average changes [%]	49.27	53.97	54.30	57.72	48.32
Average Hamming distance [%]	11.62	12.70	14.16	14.55	10.94

a particular bit of the configuration. For example, while logic functions strongly depend on bits 20 and 27, the importance of bits 0, 1, 2 and 3 is less significant for cfg8f in comparison to Fig. 6. The graphs obtained using the Hamming distance are not as uniform as in the previous case.

Table 2 summarizes average values of percentage points from Fig. 6 and Fig. 7. Both metrics suggest that the cfg8f architecture is the most sensitive to the (independent) inversions of the configuration bits. These results are also correlated with the number of unique design which can be implemented in a given reconfigurable circuit (see Table 1).

3.3 Circuit Evolution on Proposed Architectures

In order to evaluate proposed reconfigurable circuits for purposes of the evolutionary circuit design, a suitable target problem has to be chosen. As we have already mentioned, there are 8,888 functions which can be implemented on all five architectures. The goal of this experiment is to calculate the average number of generations that are needed to find all these functions on all architectures. A simple evolutionary algorithm was utilized which directly operates at the level of configuration bitstream of a particular reconfigurable circuit. It utilizes the population of five individuals. A new population is formed using (1+4) evolutionary strategy. A single bit is always inverted during mutation. The probability of selection is identical (uniform) for all bits. The fitness value is determined as the number of output bits which are correctly calculated by a candidate circuit for all possible input combinations. The maximum fitness is 32. Evolutionary algorithm was executed 200 times for every target function. A single run is stopped when a perfect fitness value is obtained or 50,000 generations are exhausted. Results are given in the first part of Table 3 which shows the average number of generations for a single function, calculated as the average number of generations from all runs and for all 8,888 target functions. We can observe that the best value is obtained for the cfg16f (797) circuit and the worst one is for the cfg8f circuit (2,734). This result corresponds with the number of unique designs. If more unique designs exist in the search space then it is more difficult to find a particular one.

3.4 Exploiting the Knowledge of Reconfigurable Architectures

Because we have recognized that logic functions are (in average) more sensitive to some configuration bits than to some others, we can speculate whether

Table 3. Summary of results for evolutionary design of 8,888 functions

Circuit	cfg4f (xornot)	cfg4f (xoror)	cfg4f (xorxnor)	cfg8f	cfg16f
Average generations (uniform mut.)	1,115	1,929	1,553	2,734	797
Average generations (nonuniform mut.)	1,009	1,728	1,480	2,710	813
Speedup	1.11	1.17	1.05	1.01	0.98

Table 4. Masks for the bits selected for nonuniform mutation

Circuit	sensitive	sensitive-inverted	average
cfg4f(xornot)	0xc410000f	0x3beffff0	0x38e80800
cfg4f(xoror)	0x0820202f	0xf7dfdfd0	0x33c01010
cfg4f(xorxnor)	0x0410101f	0xfbefefe0	0x0be80100
cfg8f	0x0810204f	0xf7efdfb0	0xf0001830
cfg16f	0x0909000f	0xf6f6fff0	0x00f02490

Table 5. Summary of results for evolutionary design of 8,888 functions when nonuniform mutation is used on selected bits

Circuit	cfg4f (xornot)	cfg4f (xoror)	cfg4f (xorxnor)	cfg8f	cfg16f
Avr. genenerations (nonunif. mut. 1/4)	1,970	3,151	2,669	3,691	1,105
Speedup	0.57	0.61	0.58	0.74	0.72
Avr. genenerations (average bits)	1,524	2,445	2,065	3,545	1,021
Speedup	0.73	0.78	0.75	0.77	0.78

a higher/lower mutation probability of these sensitive bits can improve the convergence of the evolutionary algorithm. For each reconfigurable circuit we have chosen eight the most sensitive bits and mutated them with the probability 4-times higher than other bits. This is called the nonuniform mutation in Table 3. The position of selected bits is given with respect to Figure 5 in Table 4 (column "sensitive").

We used the same evolutionary algorithm as in the previous section. Table 3 shows a small speedup of convergence (1–17 %) for four out of five investigated architectures. Therefore, it seems that our selection of sensitive bits is good.

In order to validate that the bits, which we have identified as sensitive, are really more important than other bits, two additional experiments were performed on all reconfigurable circuits. Firstly, we repeated the previous experiment, however, decreased the probability of mutation of eight the most sensitive configuration bits four times in comparison to other bits. Secondly, the previous experiment was repeated, but the probability of mutation was increased four times for 8 really average sensitive bits (their position is given as a mask in Table 4, column "average"). In both cases the speedup is much smaller than 1. That means that more generations are needed in average to find a solution and that this approach is not useful.

4 Conclusions

Although proposed circuits are very similar, significant differences were demonstrated, namely in the number of unique designs they can implement, the sensitiveness of functions to the inversions in the configuration bitstream and the average number of generations needed to find a target function. These findings are quite *unintuitive*. We believe that the proposed type of analysis can help those designers who develop new reconfigurable circuits for evolvable hardware applications. Once important (sensitive) bits of the reconfigurable circuit are identified, evolutionary algorithm can incorporate this knowledge. Additional knowledge can be included to the evolutionary algorithm and circuit architecture from the target domain. Typically, only a specific subset of all possible functions is evolved using the reconfigurable device. In this paper, we assumed that all possible functions belong to the application domain and will be evolved. Further research is needed to identify a suitable probability of mutation for the sensitive bits of a particular reconfigurable circuit and other parameters of the evolutionary algorithm.

Acknowledgements. This work was supported by the Grant Agency of the Czech Republic No. 102/06/0599 and the Research Plan No. MSM 0021630528.

References

[1] Higuchi, T., et al.: Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In: Proc. of the 2nd International Conference on Simulated Adaptive Behaviour, pp. 417–424. MIT Press, Cambridge (1993)

[2] de Garis, H.: Evolvable hardware – genetic programming of a darwin. In: Int. Conf. on Artificial Neural Networks and Genetic Algorithms, Innsbruck, Springer, Heidelberg (1993)

[3] Upegui, A., Sanchez, E.: Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs. In: The 1st NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2006), pp. 153–160. IEEE Computer Society, Los Alamitos (2006)

[4] Thompson, A., Layzell, P., Zebulum, S.: Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution. IEEE Transactions on Evolutionary Computation 3(3), 167–196 (1999)

[5] Sekanina, L.: Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. In: Tyrrell, A.M., Haddow, P.C., Torresen, J. (eds.) ICES 2003. LNCS, vol. 2606, pp. 186–197. Springer, Heidelberg (2003)