Computing and Informatics, Vol. 23, 2004, 461–486

AN EVOLVABLE COMBINATIONAL UNIT FOR FPGAS

Lukáš Sekanina, Štěpán Friedl

Faculty of Information Technology
Brno University of Technology
Božetěchova 2
612 66 Brno, Czech Republic
e-mail: sekanina@fit.vutbr.cz, friedl@liberouter.org

Manuscript received 22 December 2004

Abstract. A complete hardware implementation of an evolvable combinational unit for FPGAs is presented. The proposed combinational unit consisting of a virtual reconfigurable circuit and evolutionary algorithm was described in VHDL independently of a target platform, i.e. as a soft IP core, and realized in the COMBO6 card. In many cases the unit is able to evolve (i.e. to design) the required function automatically and autonomously, in a few seconds, only on the basis of interactions with an environment. A number of circuits were successfully evolved directly in the FPGA, in particular, 3-bit multipliers, adders, multiplexers and parity encoders. The evolvable unit was also tested in a simulated dynamic environment and used to design various circuits specified by randomly generated truth tables.

Keywords: Combinational circuit, evolutionary design, evolvable hardware, field programmable gate array

1 INTRODUCTION

Digital reconfigurable systems typically implemented using field programmable gate arrays (FPGA) are well suited for various industrial applications, including signal processing, implementation of encryption/decryption systems, embedded control systems and network devices such as routers. *Reconfigurable computing* is performed in order to effectively utilize available hardware resources and so to maximize the efficiency of these systems [3, 5, 10]. Reconfigurable systems usually utilize a set of

pre-compiled configurations that are sequentially uploaded into a reconfigurable device during the running time. The sequence of reconfigurations is either determined at the design time or emerges dynamically during the running time. However, it is practically impossible to establish a totally new configuration of hardware that was not considered at the design time, i.e. to adapt the system to a new environment.

The possibility of dynamic adaptation of hardware is intensively investigated in the field of *evolvable hardware* [11], which uses bio-inspired techniques, in particular evolutionary algorithms, to achieve the adaptation. By combining evolvable hardware and reconfigurable computing we can accomplish the *evolvable computing* [32].

The objective of this paper is to propose a technique allowing designers to perform evolvable computing at the level of HDL (Hardware Description Language), i.e. independently of a configuration mechanism of a given reconfigurable device. As a case study illustrating the method, an evolvable combinational unit for FPGAs will be introduced. Because the evolvable unit is described at the level of VHDL source code, i.e. also as a configuration for the FPGA, it can be uploaded into the FPGA when adaptation is needed. According to the concept of evolvable computing the unit can be replaced by some other component(s) after the evolution completes its task and, if needed, later uploaded to the FPGA again. The proposed method represents an alternative approach to evolvable hardware as the most applied scenario utilizes a reconfigurable device connected to a personal computer (PC) in which the evolutionary algorithm is carried out.

In order to demonstrate the concept, we decided to implement a relatively simple evolvable combinational unit of six inputs and six outputs. The unit is able to evolve (i.e. to design) the required function autonomously, in a few seconds, only on the basis of interactions with an environment. It could be classified as a real-time adaptation for some applications. In our case, and mainly for testing purposes, a connected PC supplies a truth table specifying the required behavior. A number of interesting circuits were evolved directly in the FPGA. The circuits are described and analyzed in Section 5. Furthermore, the evolvable unit was tested in a simulated dynamic environment.

We utilized the COMBO6 PCI card as an experimental platform. The COMBO6 developed in the Liberouter project is a PCI card primarily dedicated for a dual-stack (IPv4 and IPv6) router hardware accelerator [18]. This card offers a very high computational power (FPGA Virtex XC2V3000 by Xilinx, Inc. with more than 3 mil. equivalent gates, up to 2 GB DDR SDRAM, up to 9Mbit context addressable memory, etc.) which is suitable for evolvable hardware.

The paper is organized as follows. Section 2 surveys the concept of evolvable hardware and the role of FPGAs for evolvable hardware. The idea of the virtual reconfigurable device and its implementation are introduced in Section 3. Section 4 describes the proposed evolvable combinational unit. Experimental results are summarized in Section 5 and discussed in Section 6. Conclusions are given in Section 7.

2 A SURVEY OF RELEVANT RESEARCH

2.1 Evolvable Hardware

Evolvable hardware is an approach in which a physical hardware is created using the evolutionary algorithm [7, 11, 29, 42]. Typically, a configuration of a reconfigurable device is encoded in the chromosome of the evolutionary algorithm. The evolutionary algorithm is used to find such a configuration, which satisfies the requirements on the circuit behavior that are formulated in the fitness function. The fitness function can include behavioral as well as non-behavioral requirements (e.g. functionality vs. circuit size). As the method is stochastic, the quality of resulting circuits is not guaranteed. On the other hand, the evolutionary approach is sometimes able to produce the results that are better than the best conventional solutions known so far [15, 22, 37]. It is also able to produce a (suboptimal) solution in case that no conventional solution exists, the reconfigurable circuit is partially damaged or only a limited time or space is available for the design [14].

The hardware evolution is traditionally carried out on digital reconfigurable devices (such as programmable logic arrays (PLA) [11] or FPGAs [37]) and analog reconfigurable devices (such as field programmable analog arrays (FPAA) [6] and field programmable transistor arrays (FPTA) [34, 16]). Nowadays any suitable reconfigurable platform is utilized; for example, a reconfigurable antenna array [19], reconfigurable molecular array [39] or reconfigurable liquid crystals [9]. If the candidate circuits are evaluated directly in a reconfigurable device then the evolution can exploit real physical properties of the given platform (e.g. features of a given piece of silicon) or environment (e.g. temperature) and so surprising, novel and (usually) non-understandable solutions can be provided [14, 38]. The approach is referred to as the *intrinsic evolution*. In case of the *extrinsic evolution*, a circuit simulator is utilized instead of a physical reconfigurable hardware during the evolutionary process and only the best final solution is uploaded into the target device at the end of the evolutionary design.

In principle, evolvable hardware is beneficial and useful in two areas of engineering interest: (1) for the automatic discovery of novel solutions and (2) for the implementation of autonomous adaptive hardware devices. In case of the automatic design of novel circuits, the evolutionary algorithm is used only during the design phase. The main objective is to obtain novel implementations of electronic circuits automatically. Many innovative area-efficient, delay-efficient and energy-efficient circuits were evolved; see [15, 22, 35, 37, 29] for examples. In case of autonomous adaptive hardware devices, the evolutionary algorithm is responsible for continual adaptation of a device to the changing environment (e.g. in order to achieve a high-performance computation) or for the automatic recovery of a device after damage. The following applications represent typical examples: adaptive image compression [36], adaptive signal processing [13], functional recovery [14], online learning of artificial neural networks and robot controllers [25]. In case of the image compression only the evolvable hardware-based solution allowed high-performance and

high-quality printing for electrophotographic printers [12]. All the applications share a common feature: a suboptimal solution (i.e. not necessarily a perfect solution) is acceptable if the time constraints are met.

Conventional FPGA-based fault tolerant systems have been developed and utilized for a long time, e.g. [26, 24]. However, these systems are not able to generate totally new configurations after the damage in order to recover the functionality. Their behavior must be pre-programmed at the design time. One of the pre-programmed solutions is selected to replace the faulty configuration. On the other hand, it is assumed that the evolutionary functional recovery can reestablish functionality of the device for a much wider area of possible faults [14, 38].

2.2 The Role of FPGAs

Some (digital) evolvable systems have been implemented as application specific integrated circuits (ASICs). Typical examples are surveyed in [12]. However, these solutions are relatively expensive and not modifiable. We will be interested in the FPGA-based solutions in this paper. They represent a reasonable implementation option mainly because of their cost and flexibility. The FPGA-based implementations of evolvable hardware can be divided into two groups:

(1) The FPGA serves in the fitness calculation only. The evolutionary algorithm (which is executed on a personal computer or in a digital signal processor (DSP)) sends configuration bitstreams representing candidate circuits to the FPGA in order to obtain their fitness values. Initial experiments were carried out by Thompson who has evolved interesting novel circuits and discovered that the evolution can exploit physical properties of the electronic platform to build a solution [37].

(2) The entire evolvable system is implemented in an FPGA. As an example, we can mention Tufte's and Haddow's research in which they introduced the Complete Hardware Evolution approach where the evolutionary algorithm is implemented on the same chip as the evolving designs [40]. Their very simple system evolving a few register values is considered as a pipeline and demonstrated on the adaptive 1D signal filtering task. Perkins et al. presented a self-contained FPGA-based implementation of a spatially structured evolutionary algorithm that provided significant speedup over conventional serial processing for nonlinear filtering [27]. Hardware implementations of a system for the evolutionary design of image filters were proposed in [29, 43]. In another approach, Martin implemented a set of processors on the FPGA that evaluated the programs generated on the same FPGA [20]. These implementations require a hardware realization of the evolutionary algorithm—this area is relatively independent of evolvable hardware. Various solutions have been proposed, for example, see [33].

In case of evolvable hardware, the chromosomes are transformed to the configuration bitstreams and the configuration bitstreams are uploaded into the FPGA.

A partial reconfiguration allowing the reconfiguration of a selected area of the FPGA is very useful here. In contrast to the FPGA design with CAD tools, a designer has to know the format of the configuration data. Xilinx Inc. introduced Jbits to make this work easier [17]. However, it is not comfortable to decode usually very complex configuration bitstreams of FPGA vendors. Furthermore, most families of FPGAs can be configured only *externally* (i.e. from an external device connected to the configuration port). The *internal reconfiguration* means that a circuit placed *inside* the FPGA can configure the programmable elements of the same FPGA (which is important for evolvable hardware). Although the internal configuration access port (ICAP) has been integrated into the newest Xilinx Virtex II family [2], it is still too slow for our purposes. We will show in Section 4 that approximately 640 ns are needed to evaluate a candidate circuit. As soon as the reconfiguration time should be much shorter than the evaluation time (to make the approach reasonable), we need to reconfigure the circuit in tens of nanoseconds—but it is not possible using the existing configuration subsystems of FPGAs. In order to overcome the problem of the internal reconfiguration, we introduced the concept of the virtual reconfigurable circuit (VRC) [29], which will be utilized in this paper.

3 VRC AS A RECONFIGURABLE PLATFORM

3.1 The Concept

Virtual reconfigurable circuits were introduced for digital evolvable hardware as a new kind of rapidly reconfigurable platforms utilizing conventional FPGAs [28, 29]. When the VRC is uploaded into the FPGA then its configuration bitstream has to cause that the following units will be created in the FPGA: an array of programmable elements (PE), programmable interconnection network, configuration memory and configuration port. Figure 1 shows that the VRC is in fact a second reconfiguration layer (consisting of 8 PEs in the example) developed on the top of an FPGA in order to obtain fast reconfiguration and application-specific PEs.

A designer has an opportunity to design the VRC exactly according to requirements of a given application. In most cases the VRC takes a form of a regular two-dimensional array of programmable elements. A very efficient and successful approach—called *Cartesian genetic programming* (CGP)—has been developed for the evolutionary design on this structure [21]. The VRC is, in fact, a hardware implementation of the computational model used in CGP.

3.2 Cartesian Genetic Programming

In CGP, a reconfigurable circuit is modeled as an array of n_c (columns) $\times$ n_r (rows) programmable nodes. The number of circuit inputs n_i and outputs n_o is fixed. A node's input can be connected to the output of some element in the preceding columns or to some of the circuit inputs. A node has up to n_n inputs and a single

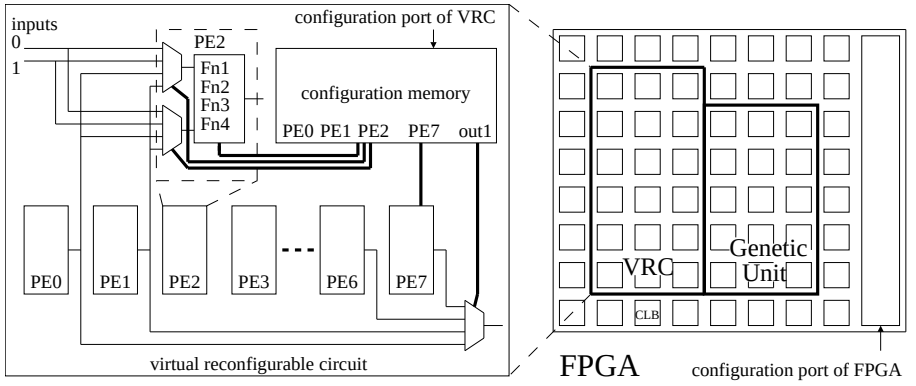

Fig. 1. Example of the internal organization of the virtual reconfigurable circuit. The programmable element PE2 is shown in detail

output. Every node is programmed to implement one of n_f functions defined in the F set. Finally, the circuit interconnectivity is defined by the *levels back parameter* L, which determines how many preceding columns of nodes may have their outputs connected to a node in the current column (the primary circuit inputs are treated in the same way as node outputs). For example, if $L = 1$, only neighboring columns may be connected; if $L = n_c$, the full connectivity is enabled. The nodes in the same column are not allowed to be connected to each other, and any node may be either connected or disconnected. In general, the circuit outputs can be taken from any node output. In this paper, feedback is not allowed and thus only the combinational circuits can be designed.

Figure 2 shows a model of a reconfigurable circuit and its corresponding configuration bitstream uploaded to establish the circuit connection. The configuration of every programmable node is represented in the chromosome using three integer values (input_1, input_2, function), which define the connection of two node inputs and the function realized in the node. The length of the chromosome measured in the genes is

$$\Lambda = n_r.n_c.(n_n + 1) + n_o. \tag{1}$$

The evolution is performed according to the following algorithm, which is a form based on a $1 + \lambda$ evolutionary strategy, where $\lambda = 4$, i.e. one parent with four offspring [1].

Algorithm 1:
(1) Randomly generate five-member initial population of circuits.
(2) Evaluate candidate circuits.
(3) Find the best circuit.
(4) Create λ mutants of the best circuit.
(5) Create a new five-member population using the mutants and the best circuit.

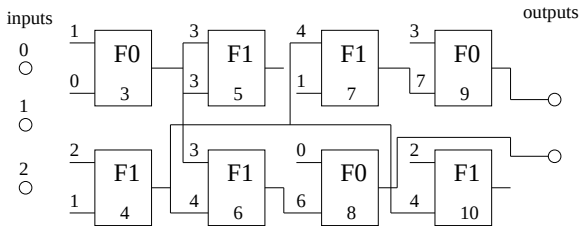

Fig. 2. Example of a circuit and its configuration in Cartesian genetic programming with the following parameters: $F = \{\text{F0, F1}\}, n_c = 4, n_r = 2, n_i = 3, n_o = 2, n_n = 2$, $n_f = 2$. The configuration information is: 1, 0, 0; 2, 1, 1; 3, 3, 1; 3, 4, 1; 4, 1, 1; 0, 6, 0; 3, 7, 0; 2, 4, 1; 9; 8. The last two integers determine the connection of the outputs

(6) Evaluate candidate circuits.
(7) If the termination criterion is not satisfied go to step (3).

3.3 Size of the Search Space

Because CGP will be implemented in hardware in Section 4.2, the following paragraphs are devoted to the summary of theoretical aspects that determine the design space. Consider that CGP, as defined in Section 3.2, is used to evolve combinational circuits.

Let a set H contain all logical functions (behaviors) of the form given by the mapping $\{0,1\}^{n_i} \to \{0,1\}^{n_o}$. The cardinality of H is

$$|H| = 2^{n_o 2^{n_i}}. \tag{2}$$

Unfortunately, independently of F (e.g. for $F = \{\text{AND, NAND, OR, NOR}\}$ as proved in [8]), almost all Boolean functions $\{0,1\}^{n_i} \to \{0,1\}$ have a circuit size complexity (the number of gates required for implementation of a particular circuit) of at least $2^{n_i} {n_i}^{-1}$. This is known as Shannon's effect [8]. However, logical circuits for key computing problems exhibit acceptable size complexity (for instance, an algorithm for multiplying two n-bit integers can be turned into a Boolean circuit of quadratic size complexity).

Assume that the outputs are fixed to the last column of PEs and not modified by evolution. In case that L-back is 1, then the number of chromosomes that form different configurations of the reconfigurable circuit (i.e. different physical circuits) is given by

$$|C| = n_f^{n_c n_r} (n_i + n_r)^{n_n n_r (n_c - 1)} n_i^{n_n n_r} \tag{3}$$

where $|C|$ is in fact the size of the search space [29].

Note that the so-called unconstrained intrinsic evolution is not considered in the previous analysis. If it were, then the evolution is allowed to utilize physical properties of the target platform and other properties of the environment (such as

changing temperature, radiation, etc.) in order to produce the required behavior. Then the circuit has to be considered as analogue and might exhibit an additional interesting functionality in comparison to a mere digital logical behavior [35, 37].

3.4 Implementation of the VRC in an FPGA

The implementation of the VRC is based on multiplexers. Figure 1 shows an example. The "virtual" PE2 depicted in Figure 1 is controlled using 6 bits determining the selection of its operands (2 + 2 bits) and its internal function (2 bits). The routing circuits are created using 4-input multiplexers. The configuration memory of the VRC is typically implemented as a register array. All bits of the configuration memory are connected to multiplexers that control the routing and selection of functions in PEs.

Because the array of PEs, routing circuits, configuration memory, style of reconfiguration and granularity of the new VRC can be designed exactly according to the requirements of a given application, designers can create an optimized application-specific reconfigurable device. Furthermore, the VRC is described in HDL, i.e. independently of a target platform. It is crucial from our perspective that the VRC can directly be connected to a hardware implementation of the evolutionary algorithm placed on the same FPGA. If the structure of the chromosome corresponds to the configuration interface of the VRC then a very fast reconfiguration can be achieved (e.g. consuming a few clock cycles only)—which is impossible by means of any other technique.

4 THE PROPOSED EVOLVABLE UNIT

This section describes a realization of the evolvable combinational unit that was completely described using VHDL and subsequently implemented on a single FPGA. A current trend in the design for FPGAs is to compose the digital systems of reusable IP cores. Since the evolvable unit is completely described in VHDL, it can be reused as an evolvable IP core. Note that the requirement for evolvable hardware at the level of IP core has been stressed by the evolvable hardware community [34, 30].

The proposed evolvable combinational unit is an application-specific system for the evolution of combinational circuits, of 6 user inputs and 6 user outputs, in a few seconds. Although these circuits are relatively small, they can be useful for the adaptive filtering [23], hashing [4] or robot controlling [25] in which large data streams have to be processed. Note that these small circuits were successfully evolved in software (extrinsically), e.g. in [22]. As seen from Figure 3, our evolvable system on a single FPGA consists of the VRC, evolutionary algorithm, circuit evaluation unit (fitness calculation) and interface to the PCI bus. The following sections describe the design considerations we made, components used, target platform and the results of synthesis.

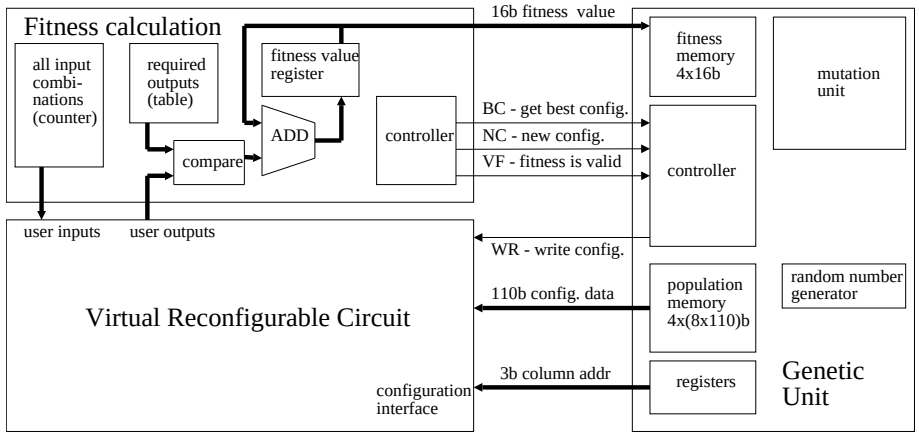

Fig. 3. Evolvable combinational unit: genetic unit, fitness calculation unit and their communication with the VRC

4.1 The Design Considerations

Based on our previous experiments [29, 31] the following CGP parameters were chosen: $n_r = 10, n_c = 8, n_i = 6, n_o = 6, n_n = 2, n_f = 8, L = 1$. These values lead to the reasonable cost of hardware implementation of the VRC and reasonable performance. It is useful that $L = 1$ because the circuit forms a natural pipeline.

It is assumed that a PE can be connected either to some of the primary inputs (6 options) or to a PE in the previous column (10 options). In total there are 16 options, i.e. four bits are needed to select the configuration of a single input of a PE. Because the PE has two inputs and other 3 bits are needed to select its function, we will need 11 bits to configure a single PE. However, only 9 bits are required for PEs in the first columns (see Figure 4).

Let us analyze the size of the search space. The number of physically different circuits is up to $|C|$ (i.e. about 10^{256}), while the number of different logical behaviors is $|H| = 2^{384} \approx 10^{116}$. In order to make the hardware implementation easier, each PE will be configured using 11 bits, although only 9 bits are needed for the first column. Therefore, the length of the chromosome is $10 \times 8 \times 11 = 880$ bits (which corresponds to the real size of the search space $\approx 10^{264}$). It is evident that the utilized reconfigurable circuit exhibits some redundancy because it is possible to show that two (or more) different configurations implement the identical behavior. However, that redundancy is beneficial for the evolutionary design [22].

The choice for 80 PEs roughly corresponds to the Shannon's effect, which indicates that about 64 PEs are needed to implement any circuit of 6 inputs and 6 outputs if no gates were shared. From the design perspective the only problem is that only neighboring PEs can be interconnected and the position of the outputs is fixed. On the other hand, the hardware implementation of the VRC is straightfor-

ward. This is a trade-off of the design since nothing is assumed about the behaviors that will be realized by the unit.

4.2 Virtual Reconfigurable Circuit

The VRC depicted in Figure 4 consists of 80 PEs equipped with flip-flops allowing pipelined processing. Each of PEs can be programmed to perform one of eight logic functions that are evident from the same figure. We recognized these functions useful during our previous research. The functions 0 ($c = a$) and 1 ($c = b$) realize a delay. Any PE can be connected to some of circuit inputs or to some of the outputs of PEs placed in the previous column. For instance, in case that the VRC is utilized for the evolution of 3×3-bit multipliers, the inputs 0-2 serve for the first operand and the inputs 3-5 serve for the second operand of the multiplier. In all cases the 6-bit output is directly connected to the middle PEs of the last column. The remaining four PEs of the last column are not utilized; however, they are implemented because the complete implementation is generated automatically (see Section 6) and their cost is negligible.

Although we restricted the search space substantially by the mentioned strategy and thus made the evolution of innovative designs probably impossible, we obtained a relatively cheap implementation in hardware utilizing only relatively inexpensive 16-input multiplexers in the interconnection network.

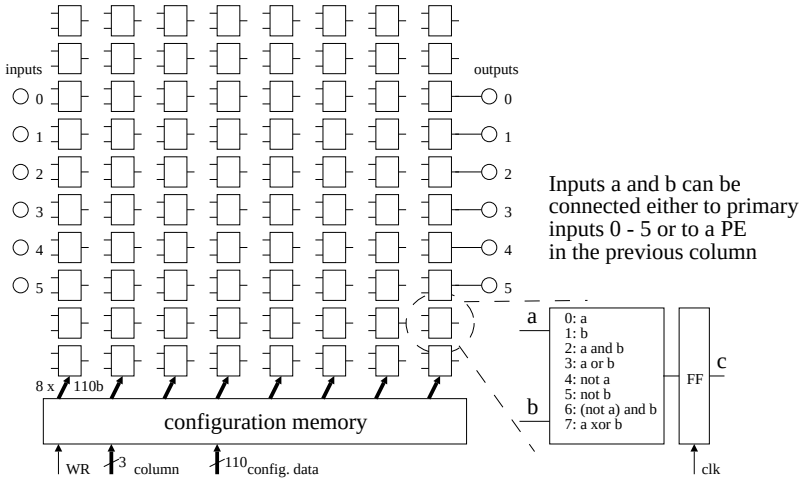

Fig. 4. The virtual reconfigurable device for evolution of 6-input/6-output combinational circuits

The 880-bit configuration bitstream is divided into eight banks of 110 bits. A single bank contains the configuration bits for a single column of the VRC. The configuration bits are stored in the 880-bit configuration register array realized using

flip-flops available in the FPGA. We need 8 clock cycles to completely change the configuration information and thus the behavior of the VRC.

4.3 Genetic Engine

The genetic unit consists of $4 \times 8 \times 110$-bit population memory, 4×16-bit fitness memory, mutation unit, comparators, multiplexers, several registers and a controller. As Figure 5 shows, it provides interface to the VRC and to the fitness calculation. The pseudo-random numbers are generated using a linear feedback shift register (LFSR), for simplicity seeded from software via the PCI bus. An analysis of LFSR for hardware evolution was performed in [20]. The author has shown that the quality of the generated pseudo-random numbers is comparable to the pseudo-random numbers generated by a cellular automaton.

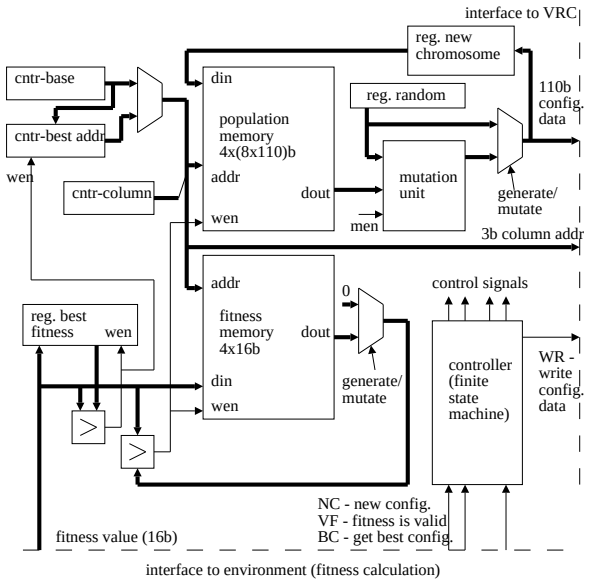

Fig. 5. Internal structure of the genetic unit

The evolvable system works as follows: First, 1024 randomly created circuits are evaluated and the four best of them are considered as the initial population. This operation is completely implemented in the FPGA. In order to make the hardware implementation easier, new populations are produced using a very simple strategy derived from the algorithm described in Section 3.2. A mutated version of each chromosome is evaluated. If the obtained fitness value is higher than the fitness value of its "parent" chromosome, then the mutated chromosome replaces the parent in the chromosome memory. This is repeated for all chromosomes in the memory until a correct solution is found or the predefined number of generations is exhausted.

Based on results of experiments, we decided to invert two bits per chromosome on average by the mutation unit. As no restrictions are applied an arbitrarily chosen bit can be inverted. The hardware implementation of the mutation is shown in Figure 6.

From the point of view of evolutionary algorithms the problem of the evolutionary circuit design is hard because the fitness landscape is not usually smooth. We are looking for a "needle in a haystack". We performed software simulations indicating that the standard crossover operators do not improve the quality of the search. Similarly to [41] we assume that a crossover operator is not useful in this case, and therefore, no crossover is implemented in hardware. We also verified that the proposed population-based search outperforms the random search.

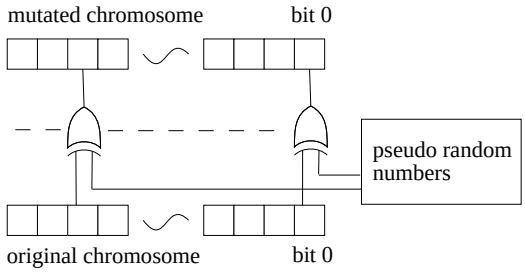

Fig. 6. Hardware implementation of mutation

4.4 Fitness Calculation

As Figure 3 shows, a special pipelined unit was designed to evaluate the circuits uploaded into the VRC. In our case, the unit generates $2^6 = 64$ test vectors (all possible input combinations are generated using a six bit counter), applies them at the VRC input, reads the output vectors from the VRC and compares them against the required vectors (that are stored in a table inside the FPGA). The fitness value is incremented for every output bit calculated correctly. Therefore, the maximal fitness value is 384. Unfortunately, the approach is not scalable because the truth table doubles by adding a single input. In the current implementation the required behavior (the truth table) is defined in software and can be changed dynamically to simulate a dynamic environment.

Thanks to pipelining, one output vector is available per a clock cycle. In the current version, the fitness value is available in $64 + 8 = 72$ clock cycles where the 8 clock cycles represent the configuration and communication overhead. Nevertheless, the overhead can be reduced to one clock cycle in case of the pipelined reconfiguration (which has not been implemented yet).

4.5 Synthesis for the COMBO6

We decided to use the COMBO6 card (see Figure 7) for our experiments, because it offers us a sufficient performance and capacity of the FPGA. Nevertheless, the primary advantage of the proposed approach is that *any* FPGA-based system of sufficient capacity can be used.

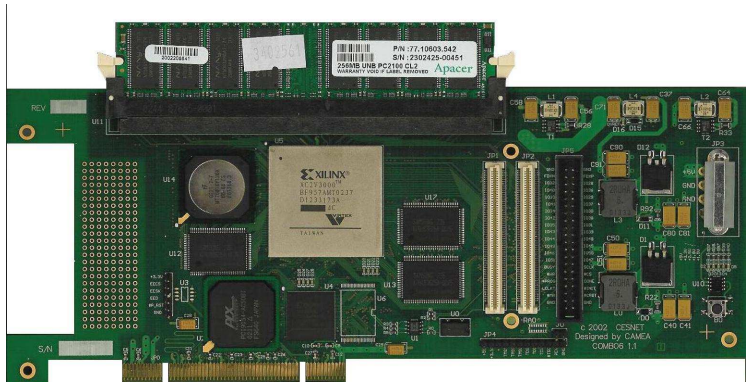

Fig. 7. The COMBO6 PCI card with Xilinx FPGA Virtex XC2V3000

The evolvable system was modeled in VHDL using ModelSim. After simulations, the design was synthesized using LeonardoSpectrum to Virtex FPGA XC2V3000bf957, which is available on the COMBO6 card. The entire evolvable system requires 470 385 equivalent gates. Table 1 summarizes the results of synthesis. For instance, while the genetic unit requires 1937 function generators and 1001 flip-flops (or latches), the VRC utilizes 2141 function generators and 940 flip-flops (or latches).

Resource	Used	Avail	Utilization
IOs	40	684	5.85%
CLB Slices	2761	14336	19.26%
Function Generators	5522	28672	19.26%
Dffs or Latches	2594	30724	8.44%
Block RAMs	5	96	5.21%
Block Multipliers	0	96	0.00%

Table 1. Results of synthesis

The population is stored in Block RAMs for this implementation. The design can operate at 77.4 MHz. The results that will be described in the next section were obtained using 50 MHz only because of easier synchronization with the PCI interface. However, there is a potential to go beyond 120MHz by optimizing some parts of the design.

5 EXPERIMENTAL RESULTS

This section summarizes the results we obtained during testing of the evolvable combinational unit. First we tried to evolve conventional combinational circuits, such as 3-bit multipliers, 3-bit adders, multiplexers and parity encoders. Then we randomly generated various behaviors (truth tables) and were interested whether a circuit can be evolved to satisfy the given requirement. Finally, we investigated the time of adaptation.

5.1 Three-bit Multiplier

The evolutionary design of the 3-bit multiplier is a traditional "benchmark" problem in the evolvable hardware literature [22, 31]. Figure 8 shows an example of the evolved 3×3-bit multiplier, which utilizes 57 PEs. This circuit was evolved after 2 450 851 generations. We performed 100 runs and obtained the fully correct solutions in all cases and in generation 4 975 829 on average. In total 125 million generations were allowed. A typical process of evolution is shown in Figure 9. Considering the average number of generations, the time of evolution is

$$t = \frac{g \cdot p(v+c)}{f_m} = \frac{4975829.4(64+8)}{50 \cdot 10^6} = 28.6\,\text{sec} \tag{4}$$

where g is the number of generations, p is the population size, v is the number of test vectors and c denotes the overhead. Considering $f_m = 100$ MHz (which we will reach with the optimized design) then we can obtain the design time 14.3 sec. on average.

It is interesting that when we change some values in the required truth table (e.g. we require $3 \times 5 = 18$, $3 \times 6 = 21$, $3 \times 7 = 24$) then the problem becomes more complicated. Although the correct solution was discovered for this modified "multiplier" in each run, the evolution required much more generations to find the solution (20 million on average). An explanation is that the modified problem does not show the symmetric truth table, while the original problem does. Section 5.5 illustrates how difficult is to evolve a circuit for a randomly generated specification (i.e. for the irregular truth table).

Table 2 summarizes all results for conventional circuits measured on the evolvable unit running at 100 MHz. These results were obtained after 100 independent runs for each problem.

5.2 Three-bit Adder

The 3-bit adder uses the two 3-bit inputs and the 4-bit output. No input carry is considered. The remaining two outputs are required to be at logic zero. We utilized the same experimental setup as in the case of multiplier. As Table 2 indicates, this problem is much easier than the design of a 3-bit multiplier. Figure 10 shows an example of the evolved adder.

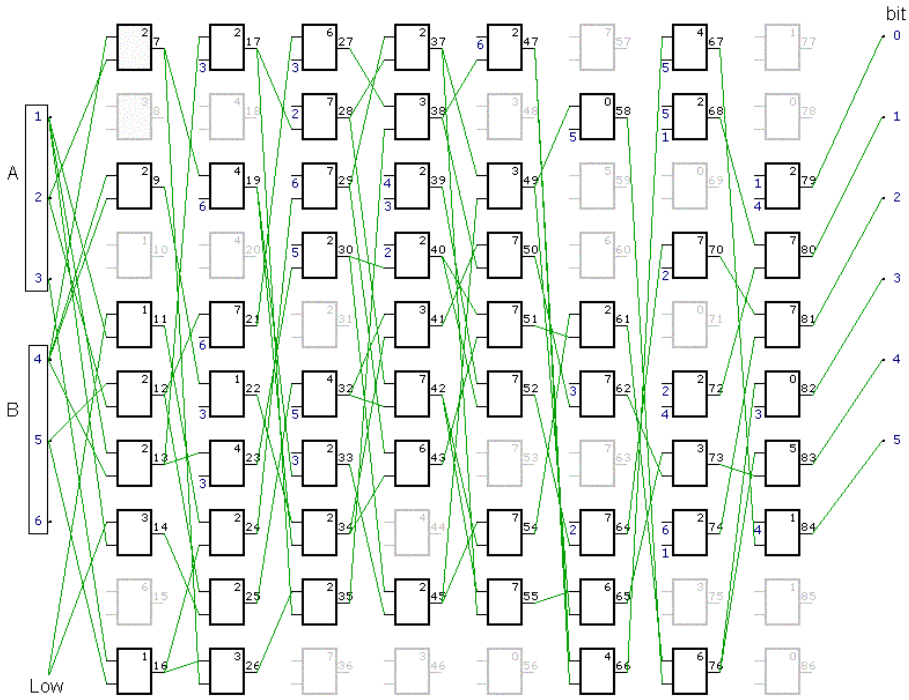

Fig. 8. A 3×3-bit pipelined multiplier evolved in the FPGA. Some PEs were not utilized and can be removed. The functions of PEs are numbered according to Figure 4

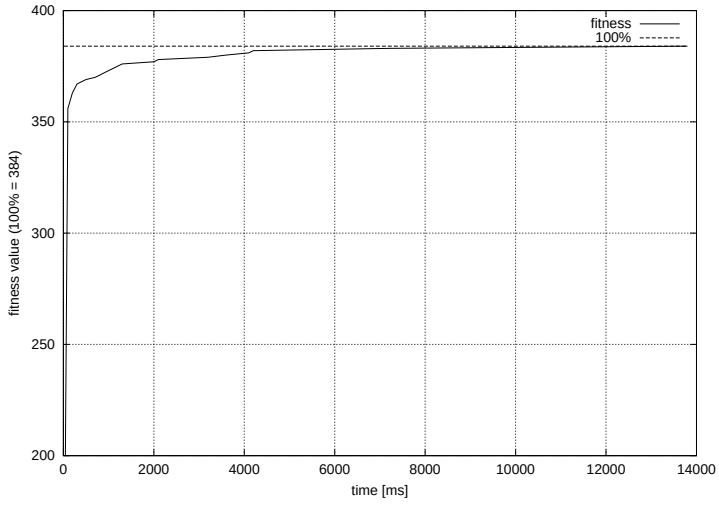

Fig. 9. A typical process of evolution of the 3-bit multiplier (at 50 MHz)

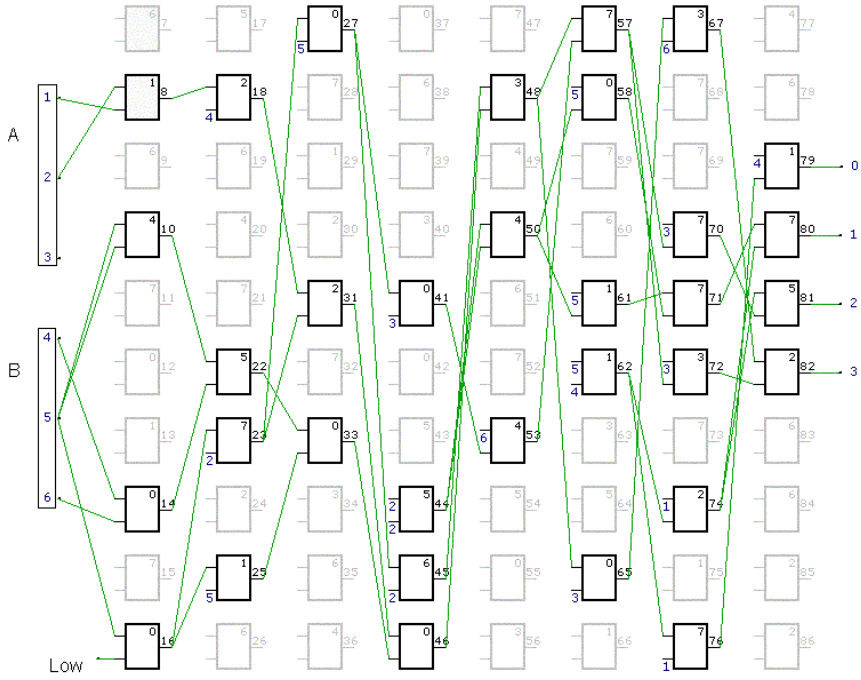

Fig. 10. A 3-bit pipelined adder evolved in the FPGA

5.3 Multiplexer

The proposed multiplexer uses four data inputs (*D0–D3*), two address inputs (*A0–A1*) and a single output. The remaining five outputs are required to be at logic zero. We utilized the same experimental setup as in the previous designs. Figure 11 shows an example of the evolved multiplexer.

5.4 Parity Encoder

The 6-input parity encoder indicates the odd occurrence of logical ones applied at the input. The remaining five outputs are required to be at logic zero. We utilized the same experimental setup as in the previous designs. Table 2 illustrates that this is the easiest problem we dealt with in the set of experiments. Figure 12 shows an example of the evolved parity encoder.

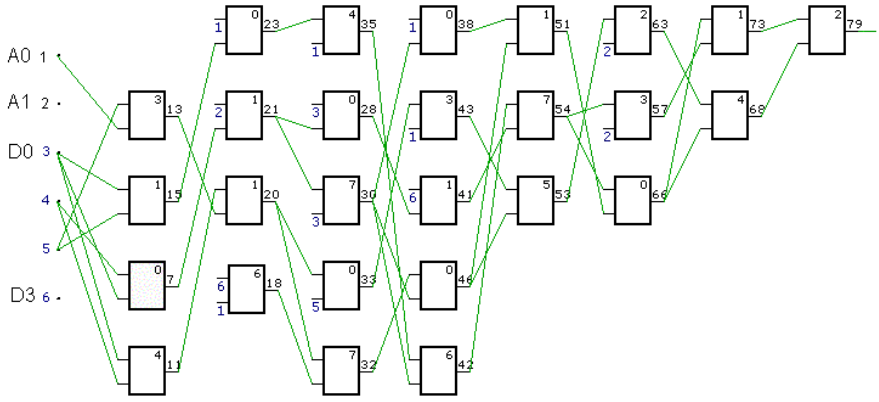

Fig. 11. A six-input multiplexer evolved in the FPGA. Only utilized PEs are shown

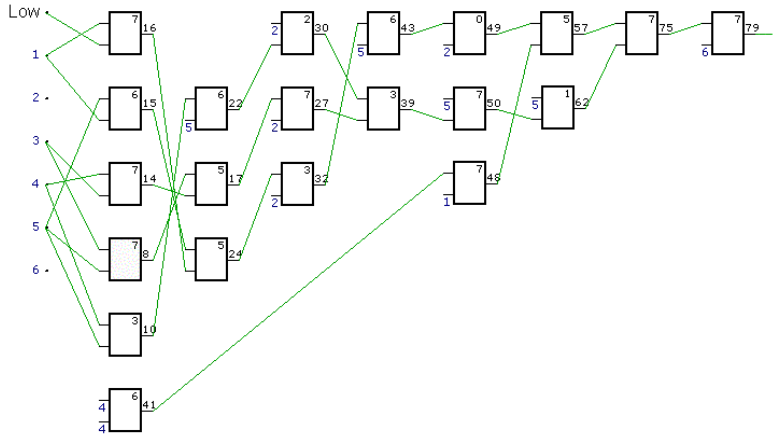

Fig. 12. A parity encoder evolved in the FPGA. Only utilized PEs are shown

5.5 Randomly Generated Functions

The previous sections have shown that the evolutionary design of small conventional circuits is not difficult and actually very fast. In order to explore the limits of the evolvable unit, we tried to evolve combinational circuits of six inputs and k outputs (where $k = 1, \ldots, 6$) whose behaviors were specified by randomly generated truth tables. For each k we randomly generated 100 behaviors (truth tables) and performed the evolutionary design of corresponding circuits, allowing 125 million generations (i.e. 6 minutes) for each run. Table 3 shows that a correct solution (i.e. the circuit with the fitness value 384) was found practically in all cases when $k = 1$

Circuit		Required generation	Fitness value	Design time [sec]
3-bit multiplier	average	4,975,829	384	14.330
	std. dev.	3,112,608	0	
	min	1,375,846	384	
	max	16,489,055	384	
3-bit adder	average	86,144	384	0.248
	std. dev.	55,037	0	
	min	13,186	384	
	max	334,053	384	
multiplexor	average	17,880	384	0.051
	std. dev.	7,600	0	
	min	4,590	384	
	max	43,601	384	
parity enc.	average	3,176	384	0.009
	std. dev.	1,775	0	
	min	555	384	
	max	9,689	384	

Table 2. The results obtained for conventional circuits: the number of needed generations, the obtained fitness values and the average design time (assuming $f_m = 100\,\mathrm{MHz}$)

or 2. No correct solution appeared for larger ks. We measured the average, minimal and maximal fitness values and the standard deviation for each set of circuits for a given k. As soon as the problem becomes harder (k increases), the average fitness value decreases. However, in the worst case ($k = 6$), the average fitness value is still 339, which corresponds to 88% of the required value.

5.6 The Adaptation Time

Although the unit is able to generate only simple combinational circuits, it could still be suitable for various applications in which the adaptation time can take several seconds and a possible non-perfect solution is acceptable.

It is assumed in these applications that the requirements on behavior of the device are changed dynamically. In our case it will be simulated by injecting some changes into the specification (i.e. the truth table will be changed dynamically). As a typical application, we can mention a robot controller in which the evolvable unit realizes a mapping of the input sensor signals to the control of motors of wheels. The mapping is changed dynamically as the robot moves in an unknown environment during the learning phase. Therefore, it is much more important to obtain a solution (perhaps suboptimal) in a reasonable time than to wait for the optimal solution for a long time. Note that the adaptation is also required in applications in which the problem specification remains unchanged, but the physical platform is changed because of damage or malfunctions. However, that is not a primary focus of this work.

Circuit		Generation	Fitness	Design time [sec]
6inp1out	average	876,334	384.00	2.524
	std. dev.	636,457	0	
	min	101,353	384.00	
	max	3,769,178	(100x) 384.00	
6inp2out	average	33,206,207	383.97	95.634
	std. dev.	24,640,136	0.17	
	min	1,688,659	383.00	
	max	116,978,551	(97x) 384.00	
6inp3out	average	75,257,854	379.44	216.743
	std. dev.	30,099,855	1.71	
	min	10,467,567	375.00	
	max	124,955,462	(1x) 383.00	
6inp4out	average	79,666,503	368.57	229.440
	std. dev.	29,168,135	2.51	
	min	14,488,953	361.00	
	max	124,523,767	(2x) 374.00	
6inp5out	average	74,457,446	354.18	214.437
	std. dev.	28,600,970	3.07	
	min	13,424,232	348.00	
	max	122,953,216	(3x) 360.00	
6inp6out	average	68,708,267	338.92	197.880
	std. dev.	30,050,336	4.14	
	min	8,579,893	330.00	
	max	124,429,008	(1x) 351.00	

Table 3. The results obtained for 100 randomly generated circuits in each set of 1–6 outputs: the number of needed generations, the obtained fitness value and the average design time (assuming $f_m = 100\,\mathrm{MHz}$, max. 125 million generations)

First, we will assume that the changes in the specification are significant. Hence the evolutionary design process will be restarted for every newly coming requirement on the circuit behavior. On the other hand, in case of the minor changes, it could be sufficient to improve only the best solution evolved so far. Therefore, in our case, the time of adaptation is given by the time of evolution starting from the completely random chromosomes.

We repeated the experiments from Section 5.5; however, the time of evolution was restricted to 0.125, 1.25, 12.5, and 125 million generations (corresponding to 0.36 sec., 3.6 sec., 36 sec. and 360 sec.) in order to investigate the time of adaptation of the evolvable unit with various k and under the time constraints. The results plotted in Figure 13 show how good circuits can be expected after the given time that is available for evolution. These values represent the averages of 100 independent runs. The corresponding standard deviations lie in the range 0–4.55.

Figure 14 shows a typical adaptation a 6 input/2 output circuit in case that only the minor changes are injected into the specification (truth table). The five bits were

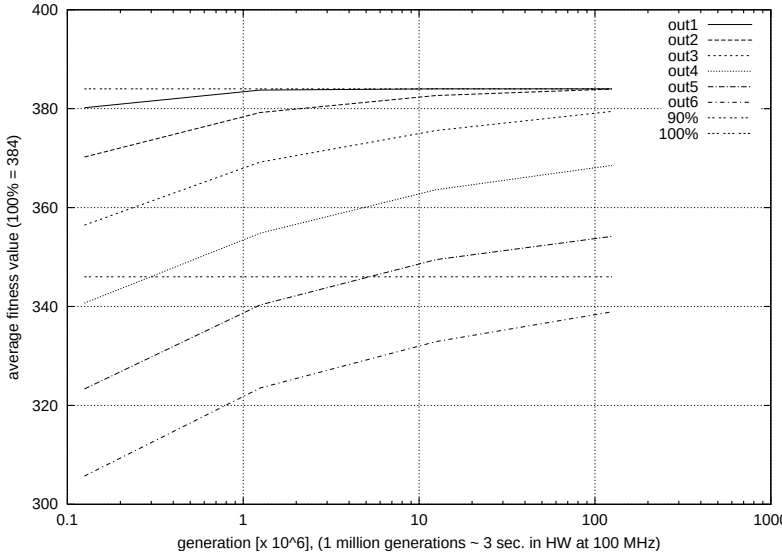

Fig. 13. The average fitness values obtained after a given time of evolution for circuits of 6 inputs and 1–6 outputs

inverted in the truth table after reaching the maximal value (384) for the previous specification. In this example, a perfect circuit was obtained after 46 seconds (on average for the unit running at 100 MHz) in all 20 cases. We can observe that very good suboptimal values are reached in less than one second.

6 DISCUSSION

All conventional circuits were evolved successfully in all runs that we performed. As the number of outputs increases it becomes harder to evolve a perfectly operating circuit in case that its behavior is described by means of a randomly generated truth table. We were not able to evolve any perfect circuit with three or more outputs although a lot of time was available for the evolution. It seems that the proposed unit is not able to do it at all. We think that the reason is twofold. First, the unit does not have sufficient resources (PEs, interconnection options, etc.). Second, the evolutionary algorithm is not efficient. However, these reasons represent the trade-off mentioned in Section 4.1. A small improvement in the performance will probably lead to more expensive hardware implementations. On the other hand, Figure 13 shows that the unit is able to produce close to perfect circuits in a few seconds (with the fitness value better than 90 %) for almost all specifications expressed by means of truth tables.

We have also to mention that it was not our goal to minimize the number of gates used in the evolved circuits. Considering evolvable adaptive systems (in

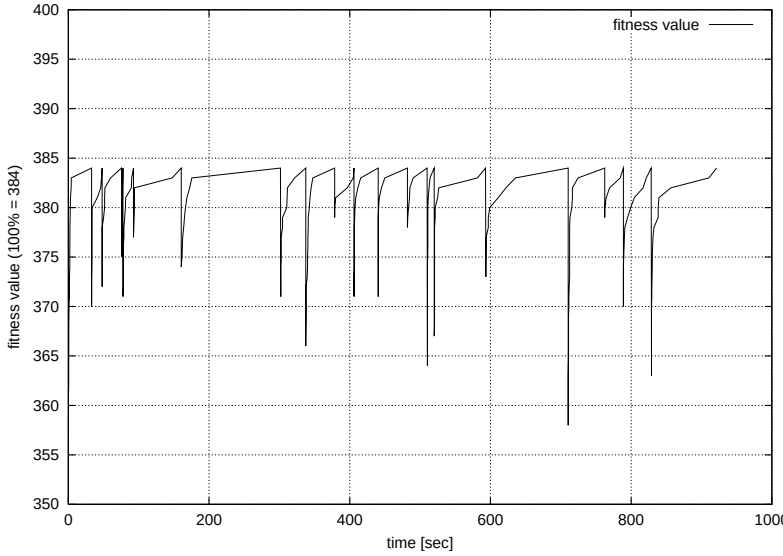

Fig. 14. A typical adaptation of a 6 input/2 output circuit in case that five bits are inverted in the specification (truth table) after reaching the maximal value (384) for the previous specification. These results are given for 100 MHz

which the evolutionary algorithm is a part of the target system), there is usually no requirement to minimize the number of elements utilized in evolved circuits. All the PEs physically exist in the system and they are available for free for the evolutionary purposes; as opposed to the strategy used in the evolutionary design of a single circuit [29]. The advantage of the evolved circuits is that they are inherently pipelined, which is useful for processing large data sets.

A software tool for the evolutionary design of combinational circuits has been developed allowing 1 million generations to be processed in 18 seconds on Pentium4 at 2.6 GHz. The proposed unit operating at 100 MHz is able to process 1 million generations in 3 seconds, i.e. the evolution is six times faster. However, the software simulator operates with 32-bit gates, so it can evaluate 32 test vectors in parallel. Considering this, the FPGA implementation is in fact 192 times faster. Note that the hardware implementation of the 32-bit reconfigurable PEs is very area-consuming in the VRC. In contrast to the evolutionary design of analog circuits using essentially slow simulators of analog circuits, the software simulator of digital CGP is very fast, so the obtained speedup is not impressive.

As far as the complete evolvable system is implemented on a part of the FPGA, this design solution is potentially suitable for adaptive embedded systems in which a common approach to evolvable hardware (i.e. the FPGA configured by a PC executing the evolutionary algorithm) can not be used or an ASIC-based approach is not an option because of higher cost.

The design of evolvable systems on FPGAs can be made easier by using CAD tools. A suite of design tools has been proposed in [31]. The basic idea behind the design system is that the user specifies the target application at a higher level of abstraction. The design system is able to automatically generate VHDL code of the application that can subsequently be synthesized for various target platforms. Hence the proposed evolvable unit can quickly be re-designed according the needs of a given application.

The proposed unit can easily be uploaded into an FPGA and so to make the FPGA evolvable. When the adaptive behavior is not required, the FPGA can be reconfigured to perform another task. The implementation cost of the unit is relatively high if compared to the size of evolved circuits. On the other hand, the approach is not suitable only for the evolution of small combinational circuits but also for the evolution of more complicated circuits at the functional level as it was shown in the area of image filtering [29]. The image filtering is a typical problem in which only a subset of all possible input combinations is tested in the fitness function.

7 CONCLUSIONS

An evolvable combinational unit was designed and implemented in the COMBO6 card. The unit is able to synthesize 6-input/6-output pipelined combinational circuits, in a few seconds, only on the basis of interactions with an environment. In particular we evolved the following circuits directly in hardware: 3-bit multipliers, 3-bit adders, multiplexers, parity encoders and a few hundreds of circuits whose behaviors were specified by randomly generated truth tables.

We have proposed a new approach to the implementation of evolvable hardware on common FPGAs and demonstrated its advantages and disadvantages on a non-trivial problem. We do believe that the method represents a step towards routine designing of evolvable systems. In fact, the problem of digital evolvable hardware design was completely transformed to the software domain by means of the proposed method since the complete design process becomes independent of a target device.

Acknowledgment

The research was performed with the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based application design methods* and No. 102/04/0737 *Modern methods of digital system synthesis*. Štěpán Friedl was supported by *6NET* project (IST-2001-32603) and the *CESNET's Programmable hardware* project.

REFERENCES

[1] Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York Oxford 1996.

[2] Blodget, B.—Roxby, J.—Keller E.—McMillan, S.—Sundararajan, P.: A Self-Reconfiguring Platform. In: Proc. of the 13th Conference on Field Programmable Logic and Applications FPL '03, Lisbon, Portugal, LNCS 2778, Springer-Verlag, 2003, pp. 565–574.

[3] Bondalapati, K.—Prasanna, V. K.: Reconfigurable Computing Systems. Proc. of the IEEE, Vol. 90, 2002, No. 7, pp. 1201–1217.

[4] Damiani, E.—Liberali, V.—Tettamanzi, A.: Evolutionary Design of Hashing Function Circuits Using an FPGA. In: Proc. of the 2nd Conference on Evolvable Systems: From Biology to Hardware, ICES '98, Lausanne, Switzerland, LNCS 1478, Springer Verlag, Berlin, 1998, pp. 36–46.

[5] Daněk, M.—Honzík, P.—Kadlec, J.—Matoušek, R.—Pohl, Z.: Reconfigurable System-on-a-Programmable-Chip Platform. In: Proc. of the 7th IEEE Design and Diagnostics of Electronic Circuits and Systems Conference, Stará Lesná, Slovakia, Institute of Informatics, Bratislava, 2004, pp. 21–28.

[6] Flockton, S. J.—Sheehan, K.: Intrinsic Circuit Evolution Using Programmable Analogue Arrays. In: Proc. of the 2nd Conf. on Evolvable Systems: From Biology to Hardware ICES98, Lausanne, Switzerland, LNCS 1478, Springer Verlag, Berlin, 1998, pp. 144–153.

[7] Gordon, T.—Bentley, P.: On Evolvable Hardware. Soft Computing in Industrial Electronics, ed by Ovaska, S., Sztandera, L. Physica-Verlag, Heidelberg 2001, pp. 279–323.

[8] Gruska, J.: Foundations of Computing. Int. Thomson Publishing Computer Press 1997.

[9] Harding, S.—Miller, J.: Evolution in Materio: Initial Experiments with Liquid Crystal. In Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, USA, IEEE Computer Society Press, 2004, pp. 298–305.

[10] Hartenstein, R.: Configware/Software Co-Design: Be Prepared for the Next Revolution. In: Proc. of the 5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, Brno, Czech Republic, 2002 Brno University of Technology, Brno 2002, pp. 19–34.

[11] 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, MIT Press, Cambridge MA 1993, pp. 417–424.

[12] Higuchi, T. et al.: Real-World Applications of Analog and Digital Evolvable Hardware. IEEE Trans. on Evolutionary Computation. Vol. 3, 1999, No. 3, pp. 220–235.

[13] Kajitani, I. et al.: A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI. In: Proc. of the 2nd Conference on Evolvable Systems: From Biology to Hardware, ICES '98, Lausanne, Switzerland, LNCS 1478, Springer Verlag, Berlin, 1998, pp. 1–12.

[14] Keymeulen, D.—Zebulum, R. S.—Duong, V.—Guo, X.—Ferguson, I.—Stoica, A.: High Temperature Experiments for Circuit Self-Recovery. In Proc. of Genetic and Evolutionary Computation Conference, GECCO 2004, Seattle, USA, LNCS 3102, Springer Verlag, Berlin, pp. 792–803.

[15] Koza, J. R.—Keane, M. A.—Streeter, M. J.: What's AI Done for Me Lately? Genetic Programming's Human-Competitive Results. IEEE Intelligent Systems, May/June 2003, pp. 25–31.

[16] Langeheine, J.—Becker, J.—Foilling, S.—Meire, K.—Schemmel, J.: A CMOS FPTA Chip for Intrinsic Hardware Evolution of Analong Electronic Circuits. In Proc. of the Third NASA/DoD workshop on Evolvable Hardware. Long Beach, CA, IEEE Computer Society, 2001, pp. 172–175.

[17] Levi, D.—Guccione, S. A.: GeneticFPGA: A Java-based Tool for Evolving Stable Circuits. In: Reconfigurable Technology: FPGAs for Computing and Applications, Proc. SPIE 3844, Bellingham, WA, 1999, pp. 114–121.

[18] Liberouter web site: `www.liberouter.org`.

[19] Linden, D. S.: Optimizing Signal Strength In-Situ Using an Evolvable Antenna System. In: Proc. of the 4th NASA/DoD Conference on Evolvable Hardware, Alexandria, Virginia, USA, 2002, IEEE Computer Society, 2002, pp. 147–151.

[20] Martin, P.: Genetic Programming in Hardware. PhD thesis, University of Essex, UK, 2003, 202 pp.

[21] Miller, J.—Thomson, P.: Cartesian Genetic Programming. In: Proc. of the 3rd European Conference on Genetic Programming, LNCS 1802, Springer Verlag, Berlin, 2000, pp. 121–132.

[22] Miller, J.—Job, D.—Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines, Vol. 1, 2000, No. 1, pp. 8–35.

[23] Miller, J.: Evolution of Digital Filters Using a Gate Array Model. In: Proc. of the Evolutionary Image Analysis, Signal Processing and Telecommunications Workshop. LNCS 1596, Springer Verlag, Berlin, 1999, pp. 121–132.

[24] Mitra, S. et al.: Reconfigurable Architecture for Autonomous Self-Repair. IEEE Design and Test of Computers. May-June 2004, pp. 228–240.

[25] Nolfi, S.—Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines, MIT Press, Cambridge MA 2000.

[26] Pradhan, D. K.: Fault-Tolerant Computer System Design. Prentice Hall 1996.

[27] Perkins, S.—Porter, R.—Harvey, N.: Everything on the Chip: A Hardware-Based Self-Contained Spatially-Structured Genetic Algorithm for Signal Processing. In: Proc. of the 3rd Int. Conf. on Evolvable Systems: From Biology to Hardware, Edinburgh, Scotland, LNCS 1801, Springer-Verlag, 2000, pp. 165–174.

[28] Sekanina, L.—Růžička, R.: Design of the Special Fast Reconfigurable Chip Using Common FPGA. In: Proc. of the 3rd IEEE Design and Diagnostics of Electronic Circuits and Systems, DDECS'00, Polygrafia SAF Bratislava, Slovakia 2000, pp. 161–168.

[29] Sekanina, L.: Evolvable Components: From Theory to Hardware Implementations. Natural Computing Series, Springer Verlag, Berlin 2003.

[30] Sekanina, L.: Towards Evolvable IP Cores for FPGAs. In: Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware, Chicago, USA, IEEE Computer Society Press, 2003, pp. 145–154.

[31] Sekanina, L.—Friedl, S.: On Routine Implementation of Virtual Evolvable Devices Using COMBO6. In Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, USA, IEEE Computer Society Press, 2004, pp. 63–70.

[32] Sekanina, L.: Evolvable Computing by Means of Evolvable Components. Natural Computing. Vol. 3, 2004, No. 3, pp. 323–355.

[33] Shackleford, B. et al.: A High-performance, Pipelined, FPGA-based Genetic Algorithm Machine. Genetic Programming and Evolvable Machines. Vol. 2, 2001, No. 1, pp. 33–60.

[34] Stoica, A. et al.: Evolution of Analog Circuits on Field Programmable Transistor Arrays. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, Palo Alto, CA, USA, 2000, IEEE Computer Society, 2000, pp. 99–108.

[35] Stoica, A.—Zebulum, R. S.—Guo, X.—Keymeulen, D.—Ferguson, M. I.—Duong, V.: Taking Evolutionary Circuit Design from Experimentation to Implementation: Some Useful Techniques and Silicon Demonstration. IEE Proceedings on Computers and Digital Techniques – Special Issue on Evolvable Hardware, Vol. 151, 2004, No. 4, pp. 295–300.

[36] Tanaka, M. et al.: Data Compression for a Digital Color Electrophotographic Printer with Evolvable Hardware. In: Proc. of the 2nd Conference on Evolvable Systems: From Biology to Hardware, ICES'98, Lausanne, Switzerland, LNCS 1478, Springer Verlag, Berlin, 1998, pp. 106–114.

[37] Thompson, A.: Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution. Distinguished Dissertation Series, Springer, London 1998.

[38] Thompson, A.—Layzell, P.—Zebulum, R. S.: Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution. IEEE Trans. on Evolutionary Computation. Vol. 3, 1999, No. 3, pp. 167–196.

[39] Tour, J. M.: Molecular Electronics. World Scientific, 2003.

[40] Tufte, G.—Haddow, P.: Prototyping a GA Pipeline for Complete Hardware Evolution. In: Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware, Pasadena, CA, IEEE Computer Society, Los Alamitos, 1999, pp. 143–150.

[41] Vassilev, V.—Miller, J.—Fogarty, T.: On the Nature of Two-Bit Multiplier Landscapes. In: Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware, Pasadena, CA, IEEE Computer Society, Los Alamitos, 1999, pp. 36–45.

[42] Yao, X.—Higuchi, T.: Promises and Challenges of Evolvable Hardware. IEEE Transactions on Systems, Man, and Cybernetics. Vol. 29, 1999, No. 1, pp. 87–97.

[43] Zhang, Y.—Smith, S.—Tyrrell, A.: Digital Circuit Design Using Intrinsic Evolvable Hardware. In Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware, Seattle, USA, IEEE Computer Society Press, 2004, pp. 55–62.

Lukáš Sekanina, Ph.D. received all his degrees from Brno University of Technology, Czech Republic. Currently he is an assistant professor at the Faculty of Information Technology, Brno University of Technology. He was a Fulbright scholar with NASA Jet Propulsion Laboratory, Pasadena (2004), a visiting lecturer with Pennsylvania State University (2001) and a visiting researcher with Department of Informatics, University of Oslo, Norway (2001). He is the author of monograph Evolvable Components (Springer Verlag, 2003) and (co)author of more than 30 refereed conference/journal papers mainly on evolvable hardware. His research interests are focused on the theory, design and hardware implementations of bio-inspired computational systems.

Štěpán Friedl received MSc degree in the electrical engineering and computer science from the Faculty of Information Technology, Brno University of Technology in 2003. Currently he is a hardware designer with the CESNET's Programmable Hardware project. His research interests are focused on the FPGA design, networking and evolvable hardware.