2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

# Extrinsic and Intrinsic Evolution of Multifunctional Combinational Modules

Lukas Sekanina, Tomas Martinek and Zbysek Gajda

***Abstract*— Multifunctional digital circuits are circuits composed of polymorphic (multifunctional) gates. In addition to its standard logic function (such as NAND), a polymorphic gate exhibits another logic function (such as NOR) which is activated under a specific condition, for example, when Vdd, temperature or illumination reaches a certain level. This paper describes the evolutionary design of multifunctional combinational circuits at the gate level using a circuit simulator and in a field programmable gate array (FPGA). The FPGA-based implementation exhibits a significant speedup against a highly optimized software simulator.**

## I. Introduction

Multifunctional digital circuits are circuits composed of *polymorphic* (multifunctional) gates. In addition to its standard logic function (such as NAND), a polymorphic gate exhibits another logic function (such as NOR) which is activated under a specific condition, for example, when Vdd, temperature or illumination reaches a certain level. Examples of polymorphic gates are available in literature [12], [14], [13]. It was shown that it is possible to consider polymorphic gates as standard building blocks and to use them, together with ordinary gates, to design multifunctional digital circuits [9]. These multifunctional circuits exhibit interesting behaviors not visible in standard digital circuits. A circuit exists that operates as an adder in case that Vdd = 3.3V and as a multiplier in case that Vdd = 1.8V [9]. Its topology is invariable; the only feature which is changed with Vdd is the functionality of some gates.

Thus, having polymorphic and ordinary gates, the task is to find a graph representing the digital circuit which performs the first desired function under first condition and the second desired function under second condition (see Fig. 1). Unfortunately, standard methods for logic synthesis are not able to solve this problem. In order to find simple multifunctional circuits at the gate level, an evolutionary approach has been utilized in the recent papers [9], [1]. The evolved circuits have up to 5 inputs and outputs and consist of a few tens of gates.

The objective of this paper is to compare two evolutionary approaches to the multifunctional combinational circuit design at the gate level:

Lukas Sekanina is with the Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic (email: sekanina@fit.vutbr.cz).
Tomas Martinek is with the Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic (email: martinto@fit.vutbr.cz).
Zbysek Gajda is with the Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic (email: gajda@fit.vutbr.cz).

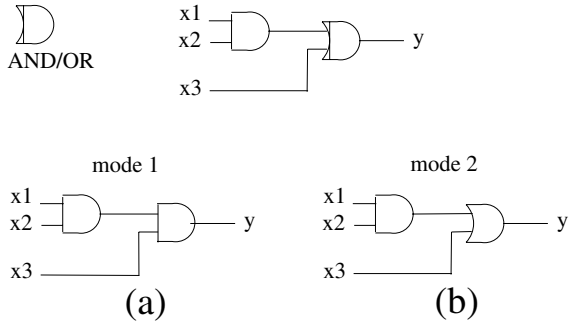


Fig. 1. Example of a polymorphic module with AND/OR gate: (a) $y = x_1.x_2.x_3$ (b) $y = x_1.x_2 + x_3$

- *Extrinsic evolution* in which a software circuit simulator is used to evaluate candidate circuits.
- *Intrinsic evolution* in which topology of a circuit is sought using a hardware accelerator. The accelerator as well as the evolution engine are implemented in a field programmable gate array.

This terminlogy (extrinsic and intrinsic evolution) is used in the field of evolvable hardware [3]. In both cases, polymorphic gates are considered as standard building blocks with properties similar to ordinary gates (i.e. they are not modeled at the transistor level). Digital multifunctional circuits will be composed of these gates. It is assumed that the FPGA-based implementation can provide a significant speedup of the design process in comparison with a highly optimized software simulator.

Section II introduces the area of polymorphic electronics. In Section III, the design problem is formulated in terms of graph theory. Section IV deals with extrinsic evolution of multifunctional combinational circuits. The problem is approached using Cartesian genetic programming that utilizes parallel simulation to accelerate the evolutionary design process. Results are given for two benchmark problems: multiplier/sorting network circuit and Plus1/Plus7 circuit. A complete FPGA implementation of multifunctional circuit evolution is proposed in Section V. Results obtained using both approaches (i.e. extrinsic and intrinsic evolution) are discussed in Section VI. Conclusions are given in Section VII.

## II. Multifunctional Digital Circuits

The concept of polymorphic electronics was proposed by Stoica et al [13]. In fact, polymorphic circuits are multifunctional circuits. The change of their behavior comes from modifications in the characteristics of components (e.g.

0-7803-9487-9/06/$20.00/©2006 IEEE

TABLE I

EXAMPLES OF EXISTING POLYMORPHIC GATES AND THEIR IMPLEMENTATION COST (# OF TRANSISTORS)

| Gate | control values | control | trans. | ref. |
|---|---|---|---|---|
| AND/OR | 27/125°C | temperature | 6 | [13] |
| AND/OR/XOR | 3.3/0.0/1.5V | ext. voltage | 10 | [13] |
| AND/OR | 3.3/0.0V | ext. voltage | 6 | [13] |
| AND/OR | 1.2/3.3V | Vdd | 8 | [14] |
| NAND/NOR | 3.3/1.8V | Vdd | 6 | [12] |

TABLE II

EXAMPLES OF MULTIFUNCTIONAL COMBINATIONAL MODULES AND THEIR IMPLEMENTATION COST (# OF GATES).

| Circuit | gates | gates used |
|---|---|---|
| 5b-parity/majority | 14 | NAND/NOR, XOR/XOR |
| 5b-parity/majority | 13 | NAND/XOR, XOR/NOR |
| 5b-boolsym/majority | 13 | NAND/NOR, XOR/XOR |
| Mult2b/sn4b | 25 | NAND/NOR, AND/AND |
| Mult2b/sn4b | 27 | $(a \vee \overline{b})$/XOR, XOR/$(a \wedge \overline{b})$ |
| 2b-mult/add | 20 | NAND/NOR, OR/XOR |
| 2b-mult/add | 23 | NAND/NOR, AND/AND |
| up/down sorting net | – | AND/OR, OR/AND |

in the transistor's operation point) involved in the circuit in response to controls such as temperature, power supply voltage, light, etc. [14], [13].

Research papers indicate many areas in which polymorphic gates could be utilized. The following list provides some examples (see a thorough analysis in [13], [14]): (a) The automatic control of power consumption when battery voltage decreases (a circuit realizes another function for lower battery voltage; however, its structure remains unchanged). (b) Implementation of a hidden function, invisible to the user, which can be activated in a specific environment (e.g. watermarking at the hardware level). (c) Intelligent sensors for biometrics, robotics, industrial measurement, etc. (d) Reverse engineering protection. (e) Implementation of low-cost adaptive systems that are able to adjust their behavior inherently.

### A. Polymorphic Gates

In theory, it could be possible to build a polymorphic gate that implements $k$ different logic functions in $k$ different environments. Practically, $k$ is 2 or 3 nowadays. If polymorphic gates were available as building blocks we could develop polymorphic electronics. The main problem is their design: All the existing polymorphic gates were discovered by means of evolutionary design techniques. It seems that a human designer is not able to accomplish this task at all.

Table I gives examples of the polymorphic gates reported in literature. For instance, the NAND/NOR gate is the most famous example [12]. The circuit consists of 6 transistors and was fabricated in a 0.5-micron CMOS technology. The circuit is stable for $\pm 10\%$ variations of Vdd and for temperatures in the range –20°–200°C. No circuits more complex than a single gate have been reported that are designed at the transistor level.

### B. Multifunctional Combinational Modules

The use of polymorphic gates as building blocks offers an opportunity to design multifunctional digital modules at the gate level. Table II surveys multifunctional combinational modules reported in [9], [1]. For example, using the polymorphic NAND/NOR gate and the standard AND gate it is possible to create a circuit which operates as a two-bit multiplier in one environment and as a two-bit adder in the second environment. Once the circuit is designed at the gate level (abstracting thus from the electric level), it does not matter whether this circuit is "reconfigured" by Vdd or a level of temperature or light.

These circuits were designed by means of evolutionary design techniques. In some cases, the circuits contain hypothetical polymorphic gates; in other cases they are composed solely of physically existing polymorphic gates. Research results indicate that it is very difficult to discover circuit topologies for nontrivial polymorphic circuits and that human designer is not able to accomplish this task because no suitable conventional design method exists. It was shown that it is useful to combine polymorphic gates with conventional gates in order to obtain compact polymorphic circuits [9].

## III. PROBLEM FORMULATION

Let $\Gamma$ be a set of polymorphic gates. Each of them is able to implement up to two functions according to a control signal which can hold two different values. A gate is in *mode j* (and so performing the $j$-th function) in the case that $j$-th value of the control signal is activated. For purposes of this paper, we will denote a polymorphic gate as $X_1/X_2$, where $X_i$ is its $i$-th logic function. For example, NAND/NOR denotes the gate operating as NAND in *mode 1* and as NOR in *mode 2*. Note that some gates can perform only one function; however, their functionality must be fully defined for each mode. For example, the conventional NAND gate considered for polymorphic circuits must perform NAND function in all modes (denoted as NAND/NAND).

A polymorphic circuit can formally be represented by graph $G = (V, E, \varphi)$, where $V$ is a set of vertices, $E$ is a set of edges between the vertices, $E = \{(a, b) | a, b \in V\}$, and $\varphi$ is a mapping assigning a function (polymorphic gate) to each vertex, $\varphi : V \rightarrow \Gamma$. As usually, $V$ models the gates and $E$ models the connections of the gates. A circuit (and also its graph) is in the *mode j* in the case that all gates are in the *mode j*.

Given $\Gamma$ and logic functions $f_1$ and $f_2$ required in different modes, the problem of the multifunctional circuit design at the gate level is formulated as follows: Find a graph $G$ representing the digital circuit which performs functions $f_1$ in its mode 1 and $f_2$ in its mode 2. Additional requirements can be specified, e.g. to minimize delay, area, power consumption etc.

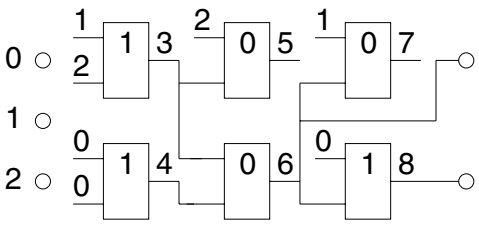


Fig. 2. An example of a 3-input circuit. CGP parameters are as follows: $L = 3$, $u = 3$, $v = 2$, $\Gamma$ = {AND (0), OR (1)}. Gates 5 and 7 are not utilized. Chromosome: 1,2,1, 0,0,1, 2,3,0, 3,4,0 1,6,0, 0,6,1, 6, 8. The last two integers indicate the outputs of the circuit.

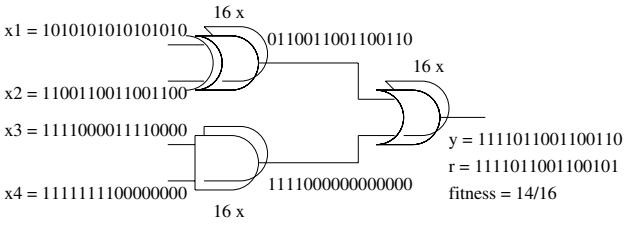


Fig. 3. Parallel simulation of a combinational circuit. $y$ is the result of simulation, $r$ is the required output

## IV. Evolutionary Design Using Simulators

Cartesian genetic programming (CGP) will be utilized to solve the problem defined in the previous section.

### A. Cartesian Genetic Programming

Miller and Thomson introduced CGP that has recently been applied by several researchers especially for the evolutionary design of combinational circuits [7], [6]. In CGP, the reconfigurable circuit is modeled as an array of $u$ (columns) $\times$ $v$ (rows) of programmable elements (gates). The number of circuit inputs, $n_i$, and outputs, $n_o$, is fixed. Feedback is not allowed. Each gate input can be connected to the output of some gate placed in the previous $L$ columns or to some of circuit inputs. The $L$ 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 gate is programmed to perform one of functions defined in the set $\Gamma$. Figure 2 shows an example and a corresponding chromosome. Every individual is encoded using $u \times v \times 3 + n_o$ integers.

The main advantage of CGP is that it uses the representation similar to real reconfigurable devices. However, the problem is that the corresponding search space is usually very rugged and thus difficult to search. Properties of CGP were analyzed in [6], [8]. Numerous experiments performed by Miller and others have shown that a very simple evolutionary algorithm achieves the highest performance. The main features of the algorithm can be summarized as: EA operates with the population of $\lambda$ individuals (typically, $\lambda = 5 - 20$). The initial population is randomly generated. Every new population consists of the best individual and its mutants. The mutation operator modifies some randomly selected genes of an individual. In case that evolution has found a solution which produces correct outputs for all possible input combinations, the number of gates is getting to minimize. Delay is not optimized. The evolution is stopped when the best fitness value stagnates or the maximum number of generations were exhausted.

In case of combinational circuit evolution, the fitness function is constructed to minimize the Hamming distance between output vectors of a candidate circuit and the required output vectors. Typically, all possible input vectors are applied to obtain the set of output vectors. This approach is tractable only for small circuits (i.e. this approach is not scalable [15]).

### B. Parallel Simulation

In a circuit simulator working at the gate level, a single gate is usually modeled using a logic function. A technique, called *parallel simulation*, was proposed to accelerate the evaluation time of a candidate circuit when the evolutionary design process is carried out in software. The idea is to utilize bitwise operators operating on multiple bits in a high-level language (such as C) to perform more than one evaluation of a gate in a single step. Therefore, when a combinational circuit under simulation has four inputs and it is possible to concurrently bitwise operations over $2^4 = 16$ bits in the simulator then this circuit can completely be simulated by applying a single 16-bit test vector at each input (see encoding in Fig. 3). In contrast, when it is impossible then sixteen four-bit test vectors must be applied sequentially. Practically, current processors allow us to operate with 64 bit operands, i.e. it is possible to evaluate the truth table of a six-input circuit by applying a single 64-bit test vector at each input. Therefore, the obtained speedup is 64 against the sequential simulation. In case that a circuit has more than 6 inputs (which is not our case) then the speedup is constant, i.e. 64. The concept of parallel simulation is utilized in CGP.

### C. The Proposed Method

In contrast to the paper [9], in which parameter $L$ was set to the maximum value (i.e. $L = u$), in this research, only neighboring columns of programmable gates may be connected, i.e. $L = 1$. This restriction is proposed in order to make hardware implementation efficient (as it will be shown in Section V). However, in order to allow interconnections of distant gates, programmable elements were equipped with the identity function (i.e. the gate can implement a wire).

Thus, the task is to find using CGP graph $G$ representing the digital circuit which performs the function $f_1$ in its mode 1 and $f_2$ in its mode 2. The proposed algorithm operates with the population of 15 individuals; every new population consists of mutants of the best individual. Only a mutation operator has been utilized that modifies one randomly selected gene of an individual. In case that the evolution has found a solution which produces the correct outputs for all possible input combinations, the number of gates is getting to minimize. Delay is not optimized. The computation is terminated in case that no improvement of

the best fitness value has been reported in a given number of recent generations (typically in 50,000 generations). The fitness value is defined as follows:

$$fitness = B_1 + B_2 + (u.v - z) \quad (1)$$

where $B_1$ (resp. $B_2$) is the number of correct output bits for $f_1$ (resp. $f_2$) obtained as response for all possible input combinations, $z$ denotes the number of gates utilized in a particular candidate circuit and $u.v$ is the total number of programmable gates available. The last term is considered only if the circuit behavior is perfect in the both modes; otherwise $uv - z = 0$.

### D. Results

In order to illustrate the performance of the proposed design tool, two circuits of four inputs and four outputs are presented here:

- 2-bit Multiplier/4-bit Sorting Network which multiplies 2-bit numbers in the first mode and sorts the input 4-bit vector in the second mode. It is a typical benchmark circuit from [9].
- 4-bit Plus1/Plus7 circuit which adds "1" to the input vector in the first mode and adds "7" to the input vector in the second mode. The circuit could by used in a counter to adaptively change its behavior.

These circuits were chosen because: (1) Their implementations exist and so it is possible to compare the obtained results with them. (2) These circuits belong to the most complex circuits that have been evolved so far, i.e. the problem is reasonably difficult.

The EA operates with the following parameters: $u = 10$, $v = 12$, $L = 1$, population size = 15 and 1 mutation is performed per chromosome. Each programmable gate can be programmed to perform one of functions from $\Gamma$ = {c = a NAND/NOR b, c = a AND b, c = a, c = b} where the last two functions denote the identity operation. After reaching the perfect functionality, the number of gates is being minimized. In total, 1M generations are produced in each run which takes 80 seconds at Athlon64 3200+ processor. CGP has utilized the parallel simulation.

Table III summarizes the obtained results. In can be seen for the Mult/SN problem that all runs were successful and 52,580 generations were needed on average to obtain a perfect circuit. Figure 4 shows the best implementation utilizing 23 gates (18 polymorphic NAND/NOR gates and 5 conventional AND gates). This circuit utilizes 74 out of 120 programmable elements; however, 51 gates are configured to operate as the identity. This implementation has 2 gates less than the best known solution [9]. Note that a conventional implementation of 2-bit multiplier requires 5 AND gates and 5 XOR gates and a conventional implementation of 4-bit sorter requires 9 AND gates and 9 OR gates.

In case of Plus1/Plus7 circuit, Table III shows that all runs were successful and 41,543 generations were needed on average to obtain a perfect circuit. The best solution utilizes 73 programmable elements out of 120 elements; however, 59 gates were configured to perform the identity function. Therefore, the circuit consists of 14 gates (6 NAND/NOR gates, 8 XOR gates).

TABLE III
EVOLUTIONARY DESIGN USING SIMULATOR

| | Mult/SN | Plus1/Plus7 |
|---|---|---|
| Runs | 10 | 10 |
| Succ. Runs | 10 | 10 |
| Generation – average | 52,580 | 41,543 |
| Generation – std. dev. | 25,364 | 44,799 |
| Gates | 23 – 31 | 14 – 23 |

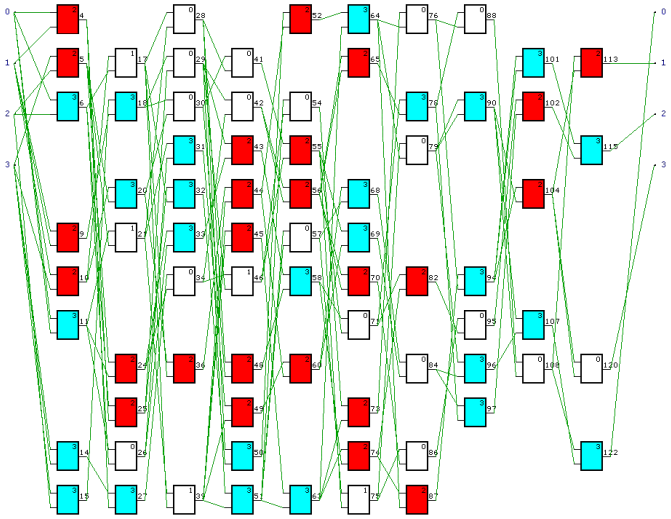

Fig. 4. The best evolved 2b-Multiplier/4b-Sorter. Functions: (0) NAND/NOR, (1) AND, (2) c = a, (3) c = b

## V. EVOLUTION IN AN FPGA

In order to speed up the evolutionary design process, a complete implementation of CGP is proposed for FPGA. Figure 5 provides overall view of the system which consists of Virtual Reconfigurable Circuit (VRC), Genetic Unit and Fitness Unit.

### A. Virtual Reconfigurable Circuit

*1) The Concept of VRC:* Most FPGA families can be configured only *externally*. 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 Xilinx Virtex II family [2], it is still too slow for our purposes. 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 internal reconfiguration, we have developed virtual reconfigurable circuits [8]. The use of VRCs has allowed us to introduce a

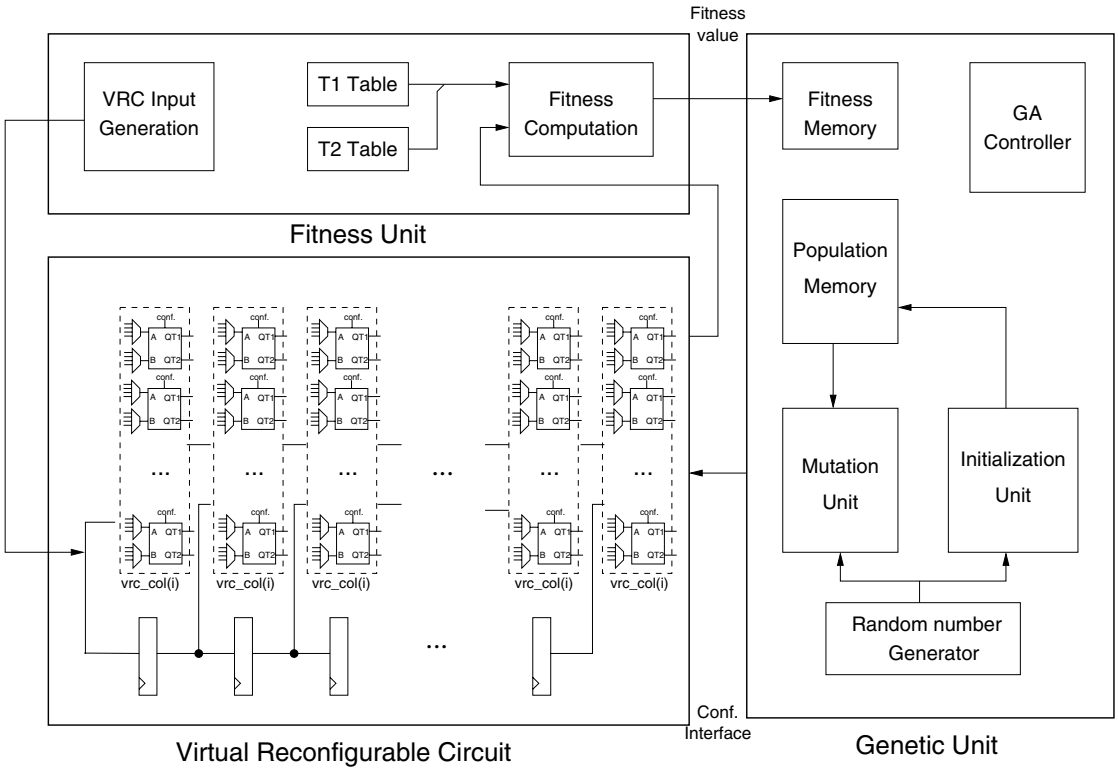


Fig. 5. Hardware implementation of multifunctional circuit evolution

novel approach to the design of complete evolvable systems in a single FPGA [8], [10], [5].

VRC is in fact a second reconfiguration layer developed on the top of an FPGA in order to obtain fast reconfiguration and application-specific programmable elements (PEs). Fig. 5 shows that VRC consists of an array of programmable elements. The routing circuits are created using 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.

*2) VRC for Multifunctional Circuit Evolution :* Figure 5 shows that VRC consists of $u \times v$ PEs. Each of them is equipped with a register to allow pipelined processing (that is possible because $L = 1$). Thus, a PE gets its inputs from the outputs of PEs placed in the previous column or from the primary inputs. The values coming from primary inputs are synchronized with the computation of PEs via a set of registers.

Figure 6 shows the implementation of a single PE. Every PE has got two outputs in order to calculate results of both modes of a polymorphic gate in one clock cycle. In order to utilize the concept of "parallel simulation" in FPGA, PEs operate with two four-bit operands and four-bit outputs. Thus, they are able to accelerate the evaluation of a candidate configuration four times. As both modes of a polymorphic gate are processed in parallel and PEs operate over four bits, we can obtain the resulting data 8 times faster than by using a naive sequential implementation. Table IV provides a list of functions implemented by PE for Plus1/Plus7 problem.

The function and connection of a single PE is defined using 10 bits; 4 bits determines the connection of its first input, 4 bits determines the connection of its second input and 2 bits are utilized to select one of four functions given in Table IV. VRC is reconfigured column by column.

TABLE IV
FUNCTIONS IN PROGRAMMABLE ELEMENTS

| Configuration | Function | Description |
|---|---|---|
| 00 | $x$ nand/nor $y$ | bitwise nand/nor |
| 01 | $x$ xor $y$ | bitwise xor |
| 10 | $x$ | $x$ to output |
| 11 | $y$ | $y$ to output |

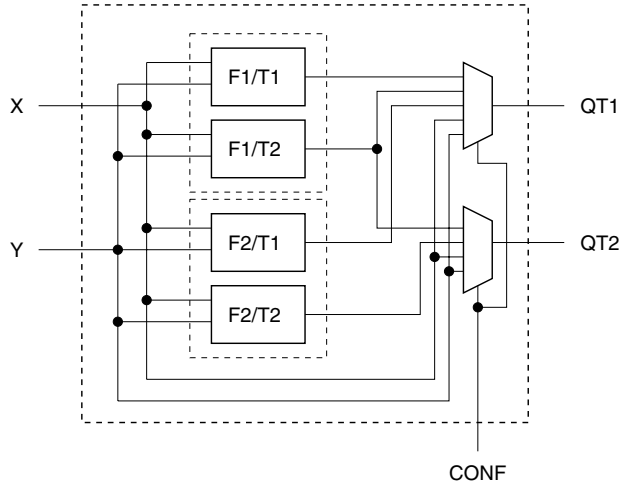


Fig. 6. Programmable element operating with 4-bit inputs and outputs and calculating both modes of a polymorphic gate in parallel

### B. Genetic Unit

Various hardware implementations of evolutionary algorithms have been proposed in the recent years, see, for example, [11]. In order to make FPGA implementation reasonably complex, the proposed EA is based only on a single mutation operator (bit inversion). The population is stored in a memory whose size is configurable. Another memory is used to store fitness values. Every new population is always generated from the best members of the previous one. EA operates in following steps:

- Initialization Unit generates the first population at random. We use a linear feedback shift register seeded from PC via PCI bus.
- Mutation Unit changes a given number of genes (bits) of a population member (this number is configurable) and the modified member is loaded into the VRC. The reconfiguration of VRC is pipelined in order to overlap the reconfiguration by useful computation. Simultaneously, the chromosome is copied into a FIFO memory.
- Genetic Unit is waiting for the evaluation performed by Fitness Unit. If the fitness value is better than the parent's fitness then the chromosome replaces its parent (the chromosome is copied from FIFO to the population memory). Otherwise, the chromosome is removed from FIFO.
- This is repeated until an appropriate number of generations are produced.

This loop is in fact divided into two parts executed concurrently. Because of the partial reconfiguration of VRC, we can send the next population member to VRC although the previous one has not been evaluated yet.

### C. Fitness Unit

Fitness unit generates input vectors for VRC and compares the outputs of VRC with the required values that are stored in table T1 (for $f_1$) and table T2 (for $f_2$). The fitness value is sent to Genetic Unit. Figure 7 shows its implementation.

### D. Target Platform and Results of Synthesis

As a target platform, COMBO6 card was utilized [4]. COMBO6 is a PCI card primarily dedicated for a dual-stack (IPv4 and IPv6) router hardware accelerator. The proposed system was described in VHDL, simulated using ModelSim and synthesized using Precision Synthesis and Xilinx ISE tools to Virtex FPGA XC2V3000bf957 chip which is available on the COMBO6 card. Host PC and COMBO6 communicate via PCI interface that allows the user to specify the values in tables T1 and T2, parameters of EA and to read configurations of VRC from FPGA. Table V summarizes results of synthesis for three different sizes of VRC.

TABLE V
RESULTS OF SYNTHESIS FOR XC2V3000

| VRC Size | Func. Gens. | Dffs | BRAMs | IOB |
|---|---|---|---|---|
| Avail | 28,672 | 28,672 | 96 | 684 |
| 8x8 | 9,178 | 2,333 | 3 | 41 |
| Util. | 32% | 8% | 3% | 5% |
| 9x10 | 13,335 | 2,862 | 6 | 41 |
| Util. | 46% | 9% | 6% | 5% |
| 12x10 | 17,614 | 3,391 | 8 | 41 |
| Util. | 61% | 11% | 8% | 5% |

The design can run at $f_{max} = 115.7$ MHz; however, we have used $f$ =50 MHz only to make easier the communication with the PLX chip running also at 50 MHz that connects COMBO6 with PC via PCI bus. The following formula determines the time of evaluation $t_{eval}$ of a candidate circuit in the proposed implementation ($u = 10, n_i = 4$):

$$t_{eval} = Max[\frac{2^{n_i}}{DW}, u].\frac{1}{f} = Max[4, 10].\frac{1}{50.10^6} = 200ns$$

where $DW$ is the number of bits which the PE operates over. Running the system at $f_{max}$ yields $t_{eval} = 86.4ns$. Time of evolution is given by formula:

$$t_e = t_{init} + g.q.t_{eval},$$

where $t_{init}$ is the initialization time (which can be neglected), $g$ is the number of generations and $q$ is the population size. Note that the configuration overhead is included into $t_{eval}$. The proposed system is able to speed up the evolution 61.7 times in comparison with the simulator discussed in Section IV-D (calculated for $f_{max} = 115.7$ MHz, $g = 1,000,000, q = 15$).

### E. Experimental Results

In order to evaluate the FPGA implementation, the evolutionary design of the Plus1/Plus7 circuit was performed 10,000 times. The evolution was stopped when the best fitness value has stagnated for 50k generations. Parameters of VRC and EA are as follows: $u = 10$, $v = 12$, configuration bitstream = 1200 bits, $\Gamma = \{$NAND/NOR, XOR, c = a, c = b$\}$, population size = 16, five mutations per chromosome. The evolution has stagnated in the generation 27,856 in average. Figure 8 shows an example of evolved Plus1/Plus7 circuit. Note that the number of utilized gates is not optimized in this implementation.

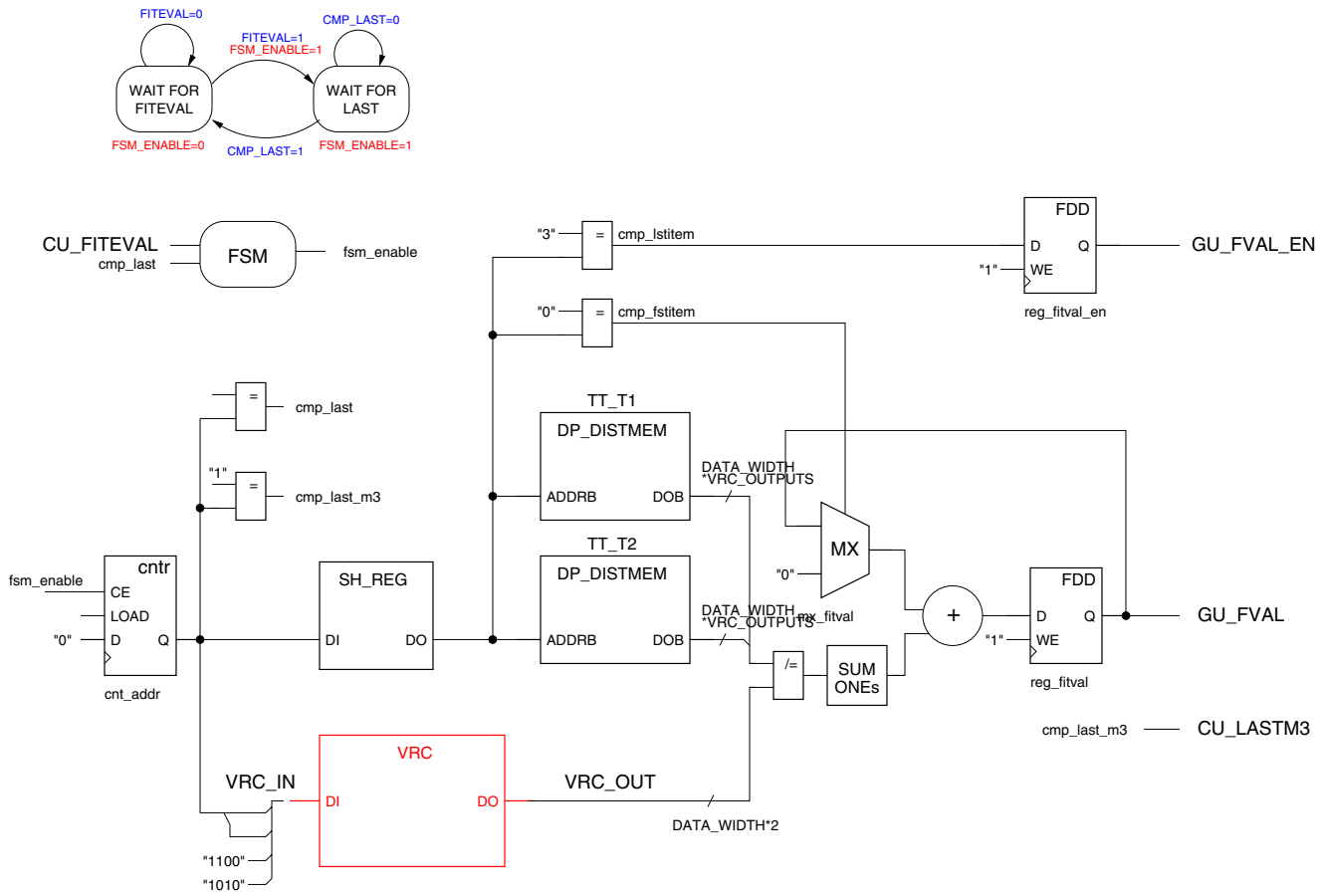


Fig. 7. Implementation of Fitness Unit

## VI. Comparison of Results and Discussion

The proposed software tool for the evolutionary design of multifunctional combinational circuits at the gate level supports the concept of parallel simulation. It enables us to accelerate the evolutionary design $b$ times if $b$ is the number of bits in operands of CPU. As the tool is written generally, it can evolve circuits of desired number of inputs and outputs, utilize desired gates and parameters of CGP. By using this tool we were able to obtain multifunctional circuits of better quality (in terms of the number of gates utilized) and in a shorter time in comparison with the results reported in [9]. In fact, in paper [9], only two out of 50 runs were successful for the Multiplier/sorting network problem although the EA utilized 128-member population and generated about 118,000 generations on average. On the other hand, the tool exhibits the same problems as other tools utilized for the evolutionary circuit design—scalability.

The FPGA-based implementation of the system has allowed speeding up the evolutionary design more than 60 times (against PC with Athlon64 3200+). This speedup is very promising if one considers that the software simulator has fully explored the potential of parallel simulation (in SW we have reached the speedup 16 against the sequential simulation). By implementing the parallel evaluation of both modes of a multifunctional circuit and four-bit operators in PEs, we have accelerated the sequential implementation eight times in FPGA. An implementation of a higher degree of parallelism (e.g. PEs operating at 16 bits, multiple VRCs etc.) requires considerably more area on the chip.

As this implementation is written generically in VHDL, the user can easily specify the desired parameters of target circuits (inputs, outputs, polymorphic gates), VRC parameters and EA parameters. A crucial problem here is finding an optimal level of parallelism and area utilization of the FPGA. Current implementation has utilized approx. 60% of Virtex XC2V3000 FPGA resources.

In contrast to the software simulator, we have recognized that a perfect solution is not produced in each run of EA in FPGA. Probably, the reason is that EA implemented in FPGA is very simple (it is, in fact, a parallel hill-climbing algorithm). In future research, we are going to implement in the FPGA the same version of EA as we have used in PC. Furthermore, in the fitness function, we will minimize the number of gates utilized in evolved circuits.

A conventional implementation of the 4-bit sorting network requires 18 gates (9 ANDs, 9 ORs), i.e. 108 transistors (a standard CMOS AND as well as OR gate requires 6 transistors). The 2-bit multiplier can be implemented using 7 gates (5 ANDs, 2 XORs), i.e. 50 transistors (XOR gate costs 10 transistors). Therefore, multiplexing the modules

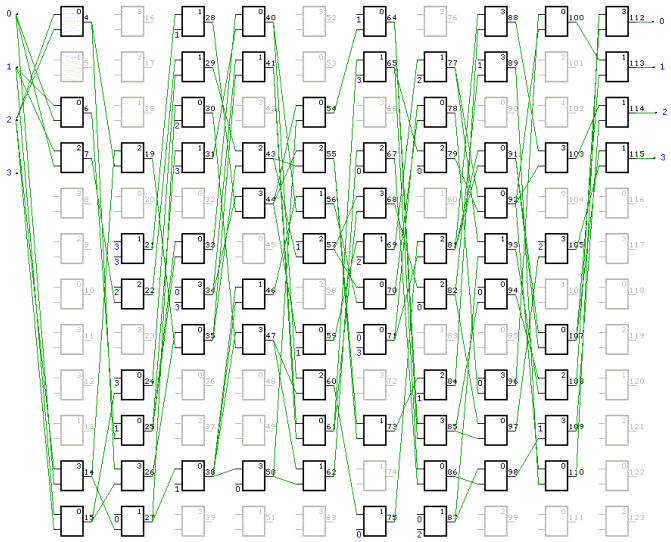

Fig. 8. VRC configured to perform Plus1/Plus7 function. Most PEs are not utilized

requires at least 158 transistors. Note that the cost of polymorphic multiplexers is not considered in this values and it is assumed that no gates can be reused in both modules. As the evolved 2-bit Multiplier/4-bit Sorting Network consists of 18 NAND/NOR gates and 5 AND gates, it costs 138 transistors, which represents a considerable reduction of the area on a chip in comparison with a potential solution based on polymorphic multiplexers.

## VII. Conclusions

Two implementations of the evolutionary design of multifunctional combinational modules were proposed and compared in this paper. Although the FPGA implementation is not so effective as the simulator (if the success rate of EA is measured), it exhibits the speedup more than 60 in comparison with the simulator. The ultimate goal of future research is to propose an FPGA-based approach to the evolutionary design of complex multifunctional circuits whose design is intractable using a common PC or a cluster of workstations. This research will also deal with the scalability issues.

## Acknowledgment

This work has been financially supported by the Grant Agency of the Czech Republic under contract No. 102/06/0599 “Methods of polymorphic digital circuit design”.

## References

[1] M. Bidlo and L. Sekanina. Providing information from the environment for growing electronic circuits through polymorphic gates. In *Genetic and Evolutionary Computation Conference (GECCO2005) workshop program*, pages 242–248, Washington, D.C., USA, 2005. ACM Press.

[2] B. Blodget, P. James-Roxby, E. Keller, S. McMillan, and P. Sundararajan. A Self-reconfiguring Platform. In *Proc. of the 13th Conf. on Field Programmable Logic and Applications FPL'03*, volume 2778 of *Lecture Notes in Computer Science*, pages 565–574, Lisbon, Portugal, 2003. Springer-Verlag.

[3] H. de Garis. An Artificial Brain: ATR's CAM-brain project aims to build/evolve an artificial brain with a million neural net modules inside a trillion cell cellular automata machine. *New Generation Computing Journal*, 12(2):215–221, 1994.

[4] Liberouter home page, 2006. http://www.liberouter.org.

[5] T. Martinek and L. Sekanina. An evolvable image filter: Experimental evaluation of a complete hardware implementation in fpga. In *Evolvable Systems: From Biology to Hardware*, volume 3637 of *LNCS*, pages 76–85. Springer Verlag, 2005.

[6] J. Miller, D. Job, and V. Vassilev. Principles in the Evolutionary Design of Digital Circuits – Part I. *Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[7] J. F. Miller and P. Thomson. Cartesian genetic programming. In *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 2000. Springer-Verlag.

[8] L. Sekanina. *Evolvable Components: From Theory to Hardware Implementations*. Natural Computing Series, Springer Verlag, 2004.

[9] L. Sekanina. Evolutionary design of gate-level polymorphic digital circuits. In *Applications of Evolutionary Computing*, volume 3449 of *LNCS*, pages 185–194, Lausanne, Switzerland, 2005. Springer Verlag.

[10] L. Sekanina and S. Friedl. An evolvable combinational unit for fpgas. *Computing and Informatics*, 23(5):461–486, 2004.

[11] B. Shackleford. A high-performance, pipelined, FPGA-based genetic algorithm machine. *Genetic Programming and Evolvable Machines*, 2(1):33–60, 2001.

[12] A. Stoica, R. Zebulum, X. Guo, D. Keymeulen, I. Ferguson, and V. Duong. Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration. *IEE Proc.-Comp. Digit. Tech.*, 151(4):295–300, 2004.

[13] A. Stoica, R. S. Zebulum, and D. Keymeulen. Polymorphic electronics. In *Proc. of Evolvable Systems: From Biology to Hardware Conference*, volume 2210 of *LNCS*, pages 291–302. Springer, 2001.

[14] A. Stoica, R. S. Zebulum, D. Keymeulen, and J. Lohn. On polymorphic circuits and their design using evolutionary algorithms. In *Proc. of IASTED International Conference on Applied Informatics AI2002*, Insbruck, Austria, 2002.

[15] V. Vassilev and J. F. Miller. Scalability problems of digital circuit evolution. In *Proc. of the 2nd NASA/DoD Workshop of Evolvable Hardware*, pages 55–64, Los Alamitos, CA, US, 2000. IEEE Computer Society.