On Routine Implementation of Virtual Evolvable Devices Using COMBO6

Lukáš Sekanina and Štěpán Friedl
Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz friedl@liberouter.org

Abstract

This paper introduces an approach showing that a complete implementation of a digital evolvable hardware system can automatically be created from a high-level specification. The approach generates the implementation of a virtual reconfigurable circuit and evolutionary algorithm independently of a target platform, i.e. as a soft IP core. The method is evaluated on the development of two high-performance evolvable systems that are utilized for fast evolutionary design of small combinational circuits, such as 3×3-bit multipliers. The COMBO6 card is employed for these experiments.

1. Introduction

Various strategies have been developed in the recent years in order to implement adaptive and evolvable hardware [10, 20]. It seems that an interesting path leads to the idea of evolvable hardware at the Intellectual Property (IP) level that has been introduced by Stoica et al. [14]. Sekanina has shown how the *evolvable IP core* can be implemented in an ordinary FPGA [11]. The realization is based on creating a virtual reconfigurable circuit (VRC) and an evolutionary algorithm in the ordinary FPGA.

Contemporary digital evolvable systems are built either as ASICs or as boards containing FPGAs combined with a powerful PC where the evolution is carried out. From this perspective, the approach utilizing a VRC offers many benefits, including: (1) It is relatively inexpensive, because the whole evolvable system is realized using an ordinary FPGA. (2) The architecture of the (virtual) reconfigurable device can be designed exactly according the needs of a given problem. (3) Because the whole evolvable system is available at the level of HDL source code, it can easily be modified and synthesized for various target platforms (FPGA families). (4) The evolvable (hardware) system can be offered and reused as software (i.e. as a soft IP core).

The objective of this paper is to demonstrate that (1) a complete implementation of a class of digital evolvable systems (which are based on the virtual reconfigurable circuits) can *automatically* be generated from a high-level specification and (2) non-trivial circuits can effectively be evolved using the implementation.

An approach is presented allowing designers to rapidly describe, simulate, synthesize and realize a domain-specific virtual reconfigurable circuit. In connection with the hardware implementation of the evolutionary algorithm, the whole evolvable system can routinely be realized in an ordinary FPGA (placed on a general-purpose board) in a reasonably short time. It will be shown that non-trivial combinational circuits (e.g. multipliers) can be evolved in a few seconds on this kind of evolvable machine. The result is similar to the evolution of analog circuits in a second on the FPTA [15]. The COMBO6 card developed in the Liberouter project is employed as a target platform [5].

The rest of this paper is organized as follows. Section 2 summarizes the approaches utilized to realize the evolvable systems on FPGAs, including virtual reconfigurable circuits. In Section 3, the proposed method for routine designing of evolvable digital systems is presented. Section 4 deals with the target platform utilized to verify the method. Section 5 describes experiments performed and their results. While the obtained results are discussed in Section 6, conclusions are given in Section 7.

2. Evolvable Systems in FPGAs

If we need to realize continually evolving systems, the evolutionary algorithm has to be placed in the target systems. The evolution algorithm is responsible for adaptation to the changing environment, which is reflected via changing fitness function.

2.1. Common Approaches

Although various (digital) evolvable systems have been implemented as ASICs (typical examples are given in [2]), this solution is relatively expensive. Hence a great effort

is invested to designing evolvable systems at the level of FPGAs. These solutions can be divided into two groups:

(1) FPGA is used for evaluation of circuits produced by evolutionary algorithm, which is executed in software (running on PC or DSP). Initial experiments were carried out by Thompson who has evolved interesting circuits and discovered that evolution can exploit physical properties of the electronic platform to build a solution [16].

(2) The whole evolvable system is implemented in the FPGA(s). This type of implementation integrates a hardware realization of evolutionary algorithm and a reconfigurable device. As an example, we can mention Tufte's and Haddow's research in which they introduced Complete Hardware Evolution approach where the evolutionary algorithm is implemented on the same FPGA as the evolving design [17]. The evolvable system is considered as a pipeline and demonstrated on adaptive 1D signal filtering [18]. In their approach only coefficients of the FIR filter (no circuits) were evolved. 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 non-linear filtering [9]. In another approach, Martin implemented a set of processors on an FPGA that evaluated (in parallel) programs generated on the same FPGA [6]. These implementations require a hardware realization of the evolutionary algorithm—this area is relatively independent of evolvable hardware. Various implementations have been proposed, for example [13].

It is a typical feature of these approaches that the chromosomes are transformed to configuration bit stream and the configuration bit stream is uploaded into the FPGA. Xilinx introduced Jbits to make this work easier [4]. Hollingworth et al. showed how Jbits can be utilized for evolvable hardware [3]. However, it is not easy to decode usually very complex configuration bit stream of FPGA vendors. Furthermore, most families of FPGAs can be configured only externally (i.e. from an external device connected to the configuration port). Internal reconfiguration means that a circuit placed *inside* an FPGA can configure 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 [1], it is still too slow for our purposes.

2.2. Virtual Reconfigurable Circuits

Evolvable systems utilizing virtual reconfigurable circuits belong to the category (2) defined in the previous section. VRCs were introduced for digital evolvable hardware as a new kind of rapidly reconfigurable platform utilizing conventional FPGAs [10, 12]. When a VRC is uploaded into the FPGA then its configuration bit stream has to cause that there will be created the following units at specified positions: array of programmable elements (PE), programmable interconnection network, configuration memory and configuration port.

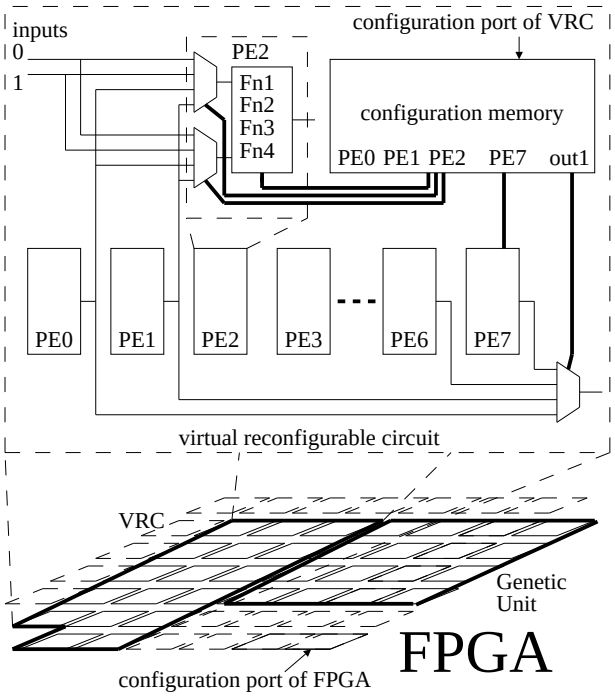

Figure 1. Example of internal organization of a VRC utilized in an evolvable system

Fig. 1 shows that the VRC is in fact a new reconfigurable circuit (consisting of 8 programmable elements in the example) realized on top of an ordinary FPGA. "Virtual" PE2 depicted in Fig. 1 is controlled using 6 bits determining selection of its operands (2+2 bits) and its internal function (2 bits). This architecture is very similar to the representation employed in Cartesian Genetic Programming (CGP) that has been developed for circuit evolution [7]. The routing circuits are created using multiplexers. The configuration memory of VRC is typically implemented as a register array. All bits of the configuration memory are connected to multiplexers that control routing and selection of functions in PEs.

The main advantage of the proposed method is that the array of PEs, the routing circuits and the configuration memory can be designed exactly according to the requirements of a given application. Furthermore, the style of reconfiguration and granularity of the new VRC can exactly fit the needs of a given application. Because VRCs can be described in HDL, they can be synthesized with various constraints and for various target platforms.

As Fig. 2 shows, the VRC can directly be connected to hardware implementation of the evolutionary algorithm

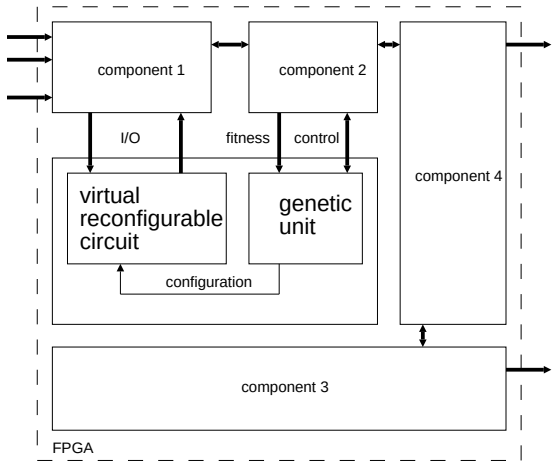

Figure 2. Evolvable system containing VRC and evolutionary algorithm

placed on the same FPGA. If the structure of chromosome corresponds to the configuration interface of VRC then a very fast reconfiguration can be achieved (e.g. consuming a few clocks only)—which is impossible by means of any other technique. In Fig. 2, the component 1 is responsible for communication with the VRC (it supplies input vectors and receives output vectors); the component 2 calculates the fitness value for a given circuit uploaded into the VRC.

Note that FPGA virtualization is sometimes utilized in the reconfigurable computing domain to increase performance. In case of evolvable hardware, Sekanina used the functional-level VRC to implement adaptive image filters [12]. However, his system was designed for a single specific application—no automatic design tools were utilized.

3. Design Approach

Figure 3 shows the general approach to routine designing of evolvable systems using virtual reconfigurable circuits. The basic idea behind the design system is that the user specifies the target application at high level of abstraction and the design system is able to automatically generate VHDL code of the application that can by synthesized for various target platforms. In particular, the specification includes: description of architecture of VRC (the number of inputs and outputs, types and organization of programmable elements, configuration options, configuration strategy, etc.), description of the evolutionary algorithm (type, parameters, chromosome encoding, etc.) and fitness evaluation, and interaction of all these units. The approach is in fact based on combining and tuning various predefined templates. Currently, the design system consists of

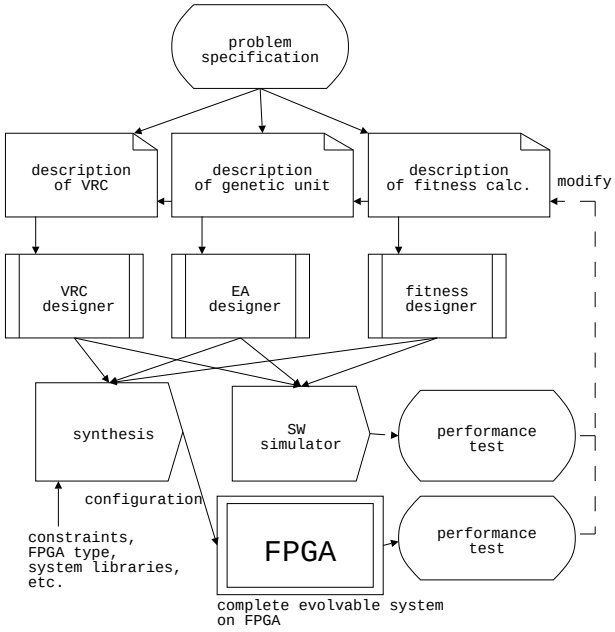

Figure 3. Design method in outline

four parts: (1) the VRC Designer, (2) the EA Designer, (3) the Fitness Designer and (4) the Integrator.

3.1. The VRC Designer

The user can choose an architecture of the virtual reconfigurable circuit and its parameters. Then this tool automatically generates synthesizable VHDL code and simulator (in C language) for the required VRC. The simulator is useful for evaluation of the VRC in software. Currently, only a single type of VRC is supported—pipelined array of programmable elements. The architecture is based on the circuit model introduced in CGP and extended to the functional level in [10]. In CGP, digital circuits are composed of programmable elements arranged in a regular pattern of x rows and y columns. The configuration bit stream determines the configuration of these elements, their interconnection and connection of primary inputs and outputs. As an example, the following code has been taken from the specification file that was prepared in order to generate VHDL code of the VRC that will be described in Section 5.2:

```
#for VHDL
ROW: 10;     #number of rows
COL: 8;      #number of columns
BIT: 1;      #datapath size (1 bit)
CFB: 80;     #number of PEs
INPUT: 6;    #number of inputs
L-BACK: {1}; #l-back
NUM-FCI: 8;  #number of functions in CFB
#number outputs {subrange} {output CFB}
```

```
OUTPUT: 6 {70..79} {72,73,74,75,76,77};
#defined functions for PE
FCE:0: {ALL}, {
    {0: VHDL: " c <= a;" },
    {1: VHDL: " c <= b;" },
    {2: VHDL: " c <= a and b;" },
    {3: VHDL: " c <= a or b;" },
    {4: VHDL: " c <= not a;" },
    {5: VHDL: " c <= not b;" },
    {6: VHDL: " c <= not (a and b);" },
    {7: VHDL: " c <= a xor b;" }
};
```

Configuration interface of all VRCs is equivalent. The templates for other types of VRCs will be available in future.

3.2. The EA Designer

The user can choose architecture of the evolutionary algorithm and its parameters. This tool then automatically generates synthesizable code of the required evolutionary algorithm. Currently, only a simple generic evolutionary algorithm is available. The genetic unit is composed of reusable parametric modules. Figure 6 shows that the designer can choose type and size of the chromosome memory (population size), chromosome size, mutation unit etc. The interfaces of all genetic units generated by the EA designer are uniform in order to ensure connectivity with the VRC. It is assumed that the unit is controlled form the environment (using signals NC, “generate new configuration”; BC, “get the best configuration”, etc.) and the environment is able to evaluate any chromosome that was generated by the genetic unit and uploaded into VRC. This strategy is known from evolvable components [10]. The templates for other types of evolutionary algorithms will be available in future.

3.3. The Fitness Designer

The environment has to evaluate any circuit uploaded into the VRC and to send the fitness value into the genetic unit (see Fig. 2). Because the fitness calculation is application-specific, its definition is left opened for the user. However, a generic template is available for typical situations, such as evaluation of combinational circuits using a truth table. The implementation is able to send test vectors to the inputs of VRC, collect responses at the outputs of VRC, compare them with the required vectors and increase the counter of fitness value. The implementation is pipelined and the pipeline naturally extends the pipeline of the VRC.

3.4. The Integrator

The integrator corresponds to the top-level-entity file in VHDL. It interconnects VRC, genetic unit and fitness calculation together. The integrator also provides user interface to the evolvable system and utilization of other resources available at the target platform. In case of COMBO6 card, it ensures communication with a personal computer via PCI bus.

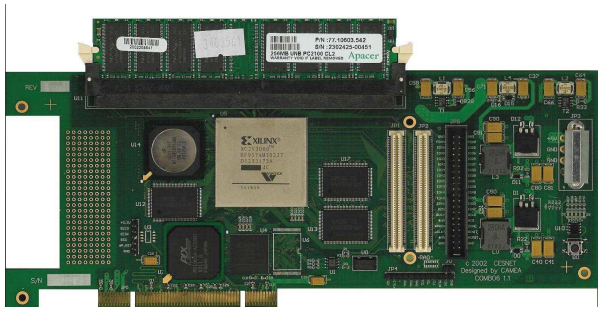

Figure 4. Combo6 card with Virtex XC2V3000

4. Target Platform: COMBO6

COMBO6 developed in the Liberouter project is a PCI card primarily dedicated for a dual-stack (IPv4 and IPv6) router hardware accelerator [5, 8]. This board offers an extremely high computational power (FPGA Virtex XC2V3000 by Xilinx, Inc. with more than 3 mil. equivalent gates, up to 2GB DDR SDRAM, up to 9Mbit context addressable memory, etc.) and so it is well suited for development and the use in various application domains, including evolvable hardware.

We decided to use this card for our experiments because it offers us a sufficient performance and capacity of FPGA. Furthermore, the card was developed in cooperation with the Faculty of Information Technology in Brno. Nevertheless, the primary advantage of the proposed approach is that any FPGA-based system of sufficient capacity can be used as the target platform.

5. Experiments and Results

The major objective of this section is to demonstrate that (1) useful digital evolvable hardware systems can be realized physically in a very short time (thus reducing the time from the problem specification till running first experiments substantially) and (2) the circuits evolved using the systems are useful and non-trivial.

5.1. Problem 1: Evolutionary Design of the 3×3-bit Multiplier

In order to demonstrate the method we decided to design a high-performance system for evolving small combi-

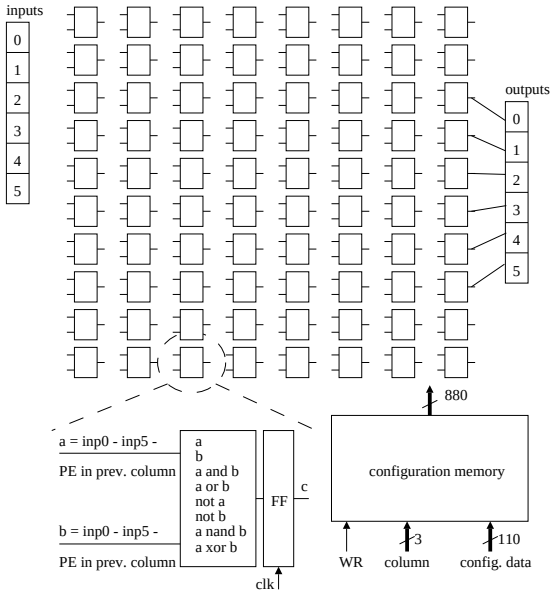

Figure 5. Virtual reconfigurable circuit generated automatically

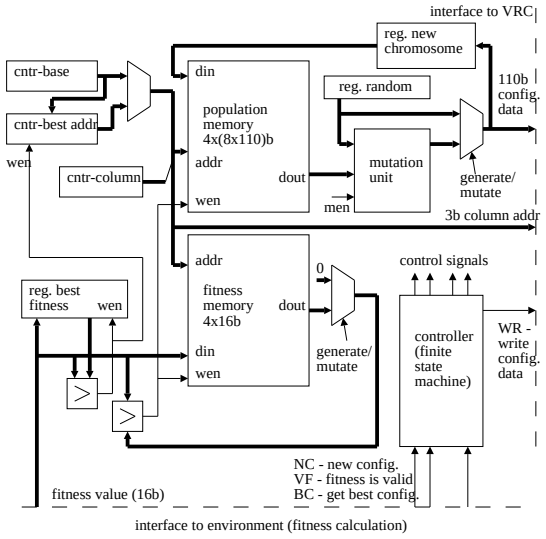

Figure 6. Genetic unit

national circuits, such as 3×3-bit multipliers, in a few seconds. These small circuits were initially evolved in software (extrinsically) by Miller et al. [7, 19]. It is believed that hardware implementation can make the evolution faster.

5.2. Proposed Evolvable System

Virtual Reconfigurable Circuit: Figure 5 shows the architecture of VRC, which was automatically generated from the specification given in Section 3.1. The circuit consists of 80 PEs (10 rows × 8 columns) equipped with flip-flops allowing pipelined processing. Each of them can be programmed to perform one of eight functions that are evident from the same figure.

Any PE can be connected to some of circuit inputs or to some of the outputs of PEs placed in the previous column. In contrary to Miller's experiments, in which inputs of PEs could be connected to a PE in whichever preceding column, we allowed the interconnection between neighboring columns only. Although we restricted the search space substantially 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.

Because the VRC is utilized for 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. The 6-bit output is directly connected to the middle PEs of the last column.

In order to define behavior of the VRC, 880 configuration bits have to be uploaded. The configuration of each PE is defined using 11 bits—the four define the connection of the first input, the other four define the connection of the second input and the remaining three bits select the function of PE. The configuration bits are stored in the 880-bit configuration register realized using flip-flops available in the FPGA. We need 8 clocks to completely change the configuration information and thus the behavior of the VRC.

Evolutionary Algorithm: Fig. 6 shows hardware realization of the genetic unit generated using the EA Designer. Chromosome memory consists of four 880-bit chromosomes; each of them is divided into eight banks per 110 bits. The initial four-member population is generated randomly and evaluated. In order to make hardware implementation easier and with respect to the results in [7], new populations are produced as follows. A mutated version of each chromosome is evaluated. If the obtained fitness value is higher than the fitness value of "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 experiments, we decided to invert four bits per chromosome on average by the mutation unit.

The controller is responsible for communication between the genetic unit and the environment and for configuring the VRC. The pseudo-random numbers are generated using LSFR seeded from software via PCI bus.

Circuit Evaluation: The Fitness Designer is able to generate a circuit evaluating the circuits uploaded in the VRC. In our case, the circuit generates $2^6 = 64$ test vectors (all possible input combinations), applies them at the VRC in-

Table 1. Results of synthesis – evolvable system in FPGA for 3×3 multiplier

Resource	Used	Avail	Utilization
IOs	41	684	5.99%
Function Generators	5207	28672	18.16%
CLB Slices	2604	14336	18.16%
Dffs or Latches	3193	30724	10.39%
Block RAMs	4	96	4.17%
Block Multipliers	0	96	0.00%

Table 2. Results of synthesis – VRC and genetic unit

Resource	VRC Used	Util.	Genetic Used	Unit Util.
IOs	127	18.57%	297	43.42%
Function Gens.	2256	7.87%	1726	6.02%
CLB Slices	1128	7.87%	863	6.02%
Dffs or Latches	940	3.06%	1177	3.83%
Block RAMs	0	0.00%	4	4.17%
Block Multipliers	0	0.00%	0	0.00%
Max freq. [MHz]	122.1		91.6	

put, reads the output vectors from VRC and compares them against the required vectors. The fitness value is incremented for every output bit calculated correctly.

It takes 8 clocks to obtain an output vector from VRC. However, thanks to pipelined processing, one output vector is available per a clock. In the current version, the fitness value is available in 64+8 = 72 clocks where the 8 clocks represent the configuration and communication overhead. Nevertheless, the overhead can be reduced in case of pipelined reconfiguration (which is has not been implemented yet).

Synthesis: After simulations in ModelSim, the design was synthesized using LeonardoSpectrum to Virtex FPGA XC2V3000bf957, which is available at COMBO6 card. The complete synthesis process took about 25 min at Sun Blade. The whole evolvable system requires 403,372 equivalent gates. Tables 1 and 2 summarize the results of synthesis.

In this implementation the population in stored in Block RAMs. The design can operate at 93.3 MHz. The results that will be described in the next section were obtained using 50MHz only because of easier synchronization with PCI interface. However, there is a potential to go beyond 120MHz by optimizing some parts of the design.

5.3. Results

Figure 7 shows an example of the evolved 3×3-bit multiplier. Our analysis has shown that the circuit utilizes 45

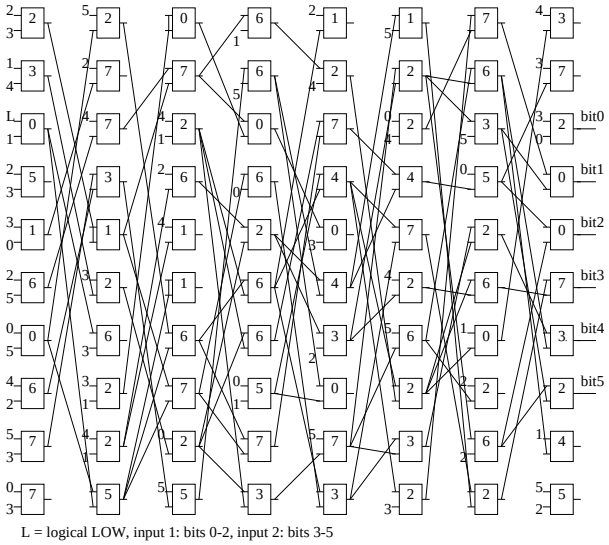

Figure 7. Evolved 3×3 multiplier

PEs. The remaining 35 PEs are not connected or can be replaced by direct wires. It is assumed that the connection can be simplified even more.

We performed 1600 runs and obtained the fully correct solutions in all cases and after generation 5,377,900 on average. The shortest run required 497,217 generations; the longest 48,804,757 generations. Considering the average number of generations, the time of evolution is

$$t = \frac{g.p(v+c)}{f_m} = \frac{5377900.4(64+8)}{50.10^6} = 31\ sec., \quad (1)$$

where g is the number of generations, p is 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 15.5 sec. on average.

5.4. Problem 2: The 4×3 Multiplier

It is easy to modify the specification file and to synthesize the evolvable system for designing other circuits, for example, 4×3 multipliers. The results of synthesis are given in Table 3. Maximal operational frequence is 89.4 MHz. Figure 8 shows an example of the evolved 4×3 pipelined multiplier. The VRC contains 10×10 PEs and utilizes $10 \times 130 = 1300$ configuration bits. Since the corresponding truth table is two times larger than in the previous case, the fitness calculation is two times slower.

We performed 19 runs and obtained the fully correct multipliers in 10 runs, after 265 millions generations on average. It corresponds to the design time about 48 minutes on average (at 50 MHz). The fastest run required 44 millions generations, i.e. 504 seconds.

Table 3. Results of synthesis – evolvable system in FPGA for 4×3 multiplier

Resource	Used	Avail	Utilization
IOs	41	684	5.99%
Function Generators	7956	28672	27.75%
CLB Slices	3978	14336	27.75%
Dffs or Latches	4702	30724	15.30%
Block RAMs	5	96	5.21%
Block Multipliers	0	96	0.00%

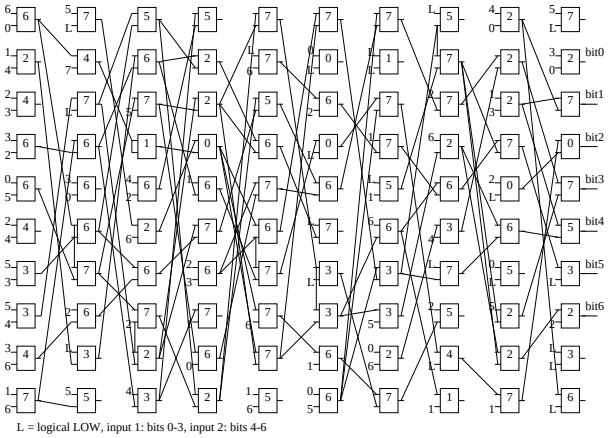

Figure 8. Evolved 4×3 multiplier

6. Discussion

The experiments have confirmed that the design time of the proposed evolvable systems is really short. Starting with specification, the design can be completed in a few hours including synthesis, placement and routing. It is easy to go back, modify some parts and synthesize a completely new evolvable system again. The main advantage is that the method is based on software approach; no operations with a physical hardware are needed if a card or a board containing a sufficiently large FPGA is available.

It is impossible to directly compare the presented results and the results from [19], for instance, because of different interconnection strategy of PEs. Recall that the best 3×3-bit multiplier evolved in [19] consists of 23 gates and the evolution was carried out using an array of 1×35 PEs, by allowing unrestricted interconnections of the PEs. In order to perform 10,000 generations, about 2 sec. are needed on Pentium@200MHz [19] (our estimate is 0.8 sec. for current Pentium IV@2.6GHz). In some cases a few millions of generations were produced to find a solution.

The speedup obtained by means of COMBO6 (f_m=100 MHz) and calculated using the available values is 69 (against Pentium) and 28 (against Pentium IV). Note that we do not compare the total time of evolution here. Table 1 indicates that four VRCs could be implemented on the same FPGA, allowing four times higher performance.

Nevertheless, in this view, the obtained speed up could be understood as low. The reason is that it is very easy to evaluate candidate circuits in software (if proper encoding is utilized). Note that it is not the case of evolving analog circuits for which the circuit simulator is usually very slow. Hence hardware implementation (JPL's FPTA, for instance) can reduce the time of the evolutionary design of analog circuits by 4+ orders of magnitude if compared with PSPICE running at Pentium II 3000 Pro [15]. The proposed approach utilizing VRCs seems to be useful for such designs in which the circuit evaluation is very time consuming in software. As a typical example, we can mention the evolutionary design of image filters [10].

The evolvable system proposed in Section 5 is devoted for speeding up the evolutionary design of small combinational circuits. However, its implementation is very area-demanding in comparison to the size of evolved circuits. The approach is much more suitable for evolvable hardware at the functional level, in which programmable elements operate with more complicated functions (such as addition, minimum, maximum, etc.) and over words instead of bits. Note that in order to support the functional level evolution, only datapath size (BIT parameter) has to be modified in the code given in Section 3.1. Therefore, it seems to be reasonable to apply the approach in real-world systems in which the evolution is responsible for adaptation.

We have to also mention that it was not our goal to minimize the number of gates used in the evolved circuits. Considering evolvable adaptive systems (in which the evolutionary algorithm is a part of the target system), there is not usually the 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 [10]. The advantage of the evolved multipliers is that they are inherently pipelined, which is useful for processing large data sets.

7. Conclusions

An approach to routine designing of high-performance evolvable systems has been introduced in this paper. Using the proposed method and tools we were able to quickly design complete evolvable systems in a physical FPGA. The design time was reduced drastically in comparison to previous approaches. The created systems were utilized to evolve small combinational circuits in a very short time. In particular we evolved the pipelined multipliers.

We do believe that the proposed approach 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. The future work will be devoted to extending the design tools in order to generate other types of virtual reconfigurable circuits and evolutionary algorithms automatically. We will work also on improving the quality of evolvable systems (hardware) generated using the tools.

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 the Research intention MSM 262200012 – *Research in information and control systems*. Štěp´an Friedl was supported by *6NET* project (IST-2001-32603) and the *CESNET's Programmable hardware* project.

References

[1] 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.

[2] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, and N. Otsu. Real-World Applications of Analog and Digital Evolvable Hardware. *IEEE Transactions on Evolutionary Computation*, 3(3):220–235, 1999.

[3] G. Hollingworth, S. Smith, and A. Tyrrell. The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic. In *Proc. of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00*, volume 1801 of *Lecture Notes in Computer Science*, pages 72–79, Edinburgh, Scotland, UK, 2000. Springer-Verlag.

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

[5] Liberouter home page, 2004. http://www.liberouter.org.

[6] P. Martin. A Hardware Implementation of a Genetic Programming System Using FPGAs and Handel-C. *Genetic Programming and Evolvable Machines*, 2(4):317–343, 2001.

[7] 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.

[8] J. Novotny, O. Fucik, and D. Antos. Project of IPv6 Router with FPGA Hardware Accelerator. In *Proc. of the 13th Conf. on Field Programmable Logic and Applications FPL'03*, volume 2778 of *Lecture Notes in Computer Science*, pages 964–967, Lisbon, Portugal, 2003. Springer-Verlag.

[9] S. Perkins, R. Porter, and N. Harvey. Everything on the Chip: A Hardware-Based Self-Contained Spatially-Structured Genetic Algorithm for Signal Processing. In *Proc. of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00*, volume 1801 of *Lecture Notes in Computer Science*, pages 165–174, Edinburgh, Scotland, UK, 2000. Springer-Verlag.

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

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

[12] L. Sekanina. Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. In *Proc. of the 5th International Conference on Evolvable Systems: From Biology to Hardware ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 186–197, Trondheim, Norway, 2003. Springer-Verlag.

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

[14] A. Stoica, D. Keymeulen, A. Thakoor, T. Daud, G. Klimech, Y. Jin, R. Tawel, and V. Duong. Evolution of Analog Circuits on Field Programmable Transistor Arrays. In *Proc. of the 2000 NASA/DoD Conference on Evolvable Hardware*, pages 99–108, Palo Alta, CA, 2002. IEEE Computer Society.

[15] A. Stoica, R. S. Zebulum, D. Keymeulen, M. I. Ferguson, and X. Guo. Evolving Circuits in Seconds: Experiments with a Stand-Alone Board Level Evolvable System. In *Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 67–74, Alexandria, Virginia, 2002. IEEE Computer Society.

[16] A. Thompson. *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Springer Verlag, London, 1998.

[17] G. Tufte and P. Haddow. Prototyping a GA Pipeline for Complete Hardware Evolution. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware*, pages 143–150, Pasadena, CA, USA, 1999. IEEE Computer Society.

[18] G. Tufte and P. Haddow. Evolving an Adaptive Digital Filter. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 143–150, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[19] V. Vassilev, D. Job, and J. Miller. Towards the Automatic Design of More Efficient Digital Circuits. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 151–160, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[20] R. Zebulum, M. Pacheco, and M. Vellasco. *Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. The CRC Press International Series on Computational Intelligence, 2002.