Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware

Lukáš Sekanina

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz, phone: +420 541141215

Abstract. The paper introduces a new method for the design of real-world applications of evolvable hardware using common FPGAs (Field Programmable Gate Arrays). In order to avoid "reconfiguration problems" of current FPGAs a new *virtual reconfigurable circuit*, whose granularity and configuration schema exactly fit to requirements of a given application, is designed on the top of an ordinary FPGA. As an example, a virtual reconfigurable circuit is constructed to speed up the software model, which was utilized for the evolutionary design of image operators.

1 Introduction

In recent years, researchers have recognised the essential difference between evolutionary circuit design and evolvable hardware [1].

In case of *evolutionary circuit design* a single circuit is evolved. An evolutionary algorithm is used only during design phase and thus it plays the role of a "designer". While quality and implementation cost of the resulting circuit are main design objectives, the time needed to evolve a satisfactory circuit for a given application may not be critical. One could imagine that it could be economically profitable to evolve a single circuit for one month, provided that it can be reused for the next five years. Hence the design can be approached using a circuit simulator in software (that is, in *extrinsic evolution*).

On the other hand it can be supposed that a noticeable portion of a target (embedded) system will be implemented in hardware in case of real-world applications of *evolvable hardware* (EHW). Such applications typically operate in time-varying environments. Then EHW ensures either adaptation (e.g. in robotics) or high performance computation (e.g. in image compression), which is unreachable using conventional approaches. In contrast to evolutionary circuit design, the evolutionary algorithm is a part of a target system. However its role is rather to provide *sufficient long-time mean performance* (quality) of the evolved circuits than to find "innovative" circuits. Unlike evolutionary circuit design, the amount of resources (gates) needed to realize the final circuit is not usually important because some resources are always devoted for evolutionary purposes and may be used for free.

While there exist some ASIC-based (perhaps already commercialized) real-world applications of EHW [2], implementations of FPGA-based real-world applications of EHW remain practically unexplored. Although FPGAs are widely accessible reconfigurable circuits (RC), the following points represent obstacles for their usage in EHW: (1) In current FPGAs the reconfiguration system is not sufficient for EHW because the *partial reconfiguration* is not fully supported (the XC6200 family is off the market now and the usage of JBits is too complicated [5]) and FPGAs can only be reconfigured externally. (2) FPGAs are based on Configuration Logic Blocks (CLBs) whose granularity is too fine for evolution of complex circuits. And these complex circuits are required in real-world applications. Hence in addition to the problems mentioned previously, *scalability* problems of EHW have to be solved in real-world applications [3,4].

In this paper, we present a method for the design of real-world applications of EHW using common FPGAs. A new *virtual* RC, whose granularity and reconfiguration schema exactly fit to the requirements of a given application, will be designed on the top of a common FPGA. It will enable the implementation of a target system in a low-cost, commercial off-the-shelf FPGA and to achieve adaptation to a changing environment. The method will be illustrated in the design of the virtual RC for an evolvable image pre-processing component.

The design will be approached in two major steps: (1) Circuit software simulator will be employed during the evolutionary circuit design of some typical circuits from application domain in order to find "optimal" organization of the virtual RC and evolutionary algorithm. (2) The virtual RC and evolutionary algorithm will be designed and synthesized into an FPGA according to outcomes of step 1. Note that this paper deals only with design of the virtual RC.

Section 2 introduces the problem domain and summarizes the results we obtained from the previous research. The implementation of the virtual RC is described in Section 3. Section 4 discusses the proposed approach. And finally, conclusions are given in Section 5.

2 Evolutionary Image Operator Design

In order to ensure the adaptability of a target embedded system (e.g. to a varying type of noise), we have to find such an organization of the RC and genetic operators, which will enable us to evolve *various* image operators and filters in a reasonable time. Every image operator will be considered as digital circuit of nine 8bit inputs and a single 8bit output, which processes gray-scaled (8bits/pixel) images. As Fig. 1 shows every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbors of the processed image.

We approached the problem using Cartesian Genetic Programming (CGP) [6]. However after a number of experiments we claim that it is practically impossible to evolve a gate-level image operator of 72 input and 8 output bits, which is able to compete against conventional solutions. In order to allow the design of such complex circuits, we extended CGP to *functional level* [7]. Instead of CLBs and 1 bit connection wires, Configurable Functional Blocks (CFBs) and 8bit datapaths are utilized (see Fig. 1 and Fig. 3).

Similarly to conventional CGP, only very simple variant of the evolutionary algorithm is employed. Population size is 16. The initial population was gener-

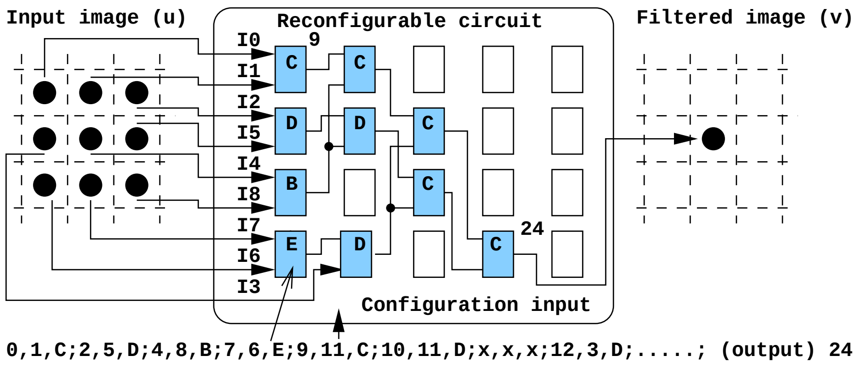

Fig. 1. A typical arrangement of an experiment for evolutionary design of image operators. A sample chromosome (circuit) is uploaded and the active CFBs are marked.

ated randomly, however only function "C" (see Fig. 3) was used in some runs. In case of evolutionary circuit design, the evolution was typically stopped (1) when no improvement of the best fitness value occurs in the last 50000 generations, or (2) after 500000 generations.

Only mutation of two randomly selected active CFBs is applied per circuit. Four individuals with the highest fitness are utilized as parents and their mutated versions build up the new population. In order to evolve a *single* filter (digital circuit), which suppresses a *given* type of noise, we need an original image to measure the fitness of the evolved filter. The generality of the evolved filters (i.e. whether the filter operates sufficiently also for other images of the same type of noise) is tested using a test set.

The design objective is to minimize the difference between the corrupted image and the original image. We chose to measure *mean difference per pixel* since it is easy for hardware implementation. Let u denote a corrupted image and let v denote a filtered image. The original (uncorrupted) version of u will be denoted as w. The image size is $K \times K$ (K=256) pixels but only the area of 254 x 254 pixels is considered because the pixel values at the borders are ignored and thus remain unfiltered. The fitness value of a candidate filter is obtained as follows: (1) the circuit simulator is configured using a candidate chromosome, (2) the circuit created is used to produce pixel values in the image v, and (3) fitness is calculated as

$$fitness = 255.(K-2)^2 - \sum_{i=1}^{K-2} \sum_{j=1}^{K-2} |v(i,j) - w(i,j)|.$$

Now we can summarize CGP parameters that lead to efficient evolutionary design: RC consists of 2-input CFBs placed in array of 10 columns and 4 rows. An input of each CFB may be connected to circuit's inputs or to the output of a CFB, which is placed in the previous two columns (i.e. CGP's L-back parameter is L=2). After statistical analysis of the functions utilized in the evolved circuits, we recognized that CFB should support functions listed in Fig. 3. Let us note that functions “C”, “E”, and “F” are the most important for successful evolution.

A number of unique image filters for Gaussian, uniform, block-uniform and shot noise was reported [7,9]. These filters typically consist of 10-25 CFBs and show better quality in comparison with typical conventional filters (such as mean and median filters). Surprisingly in some cases the number of equivalent gates needed for implementation in FPGA XC4028XLA was also lower than in case of conventional filters. The very efficient edge detectors were also evolved using the same system. Furthermore, the system was able to produce sequences of very good filters in simulated time varying environment (changing types of noise) where only 20000 populations were generated per circuit [10].

3 Virtual RC for Evolution of Image Operators

The previous section dealt with an evolutionary image operator design problem in which it was not any problem to spend 1 day with a single filter design on common PC. However if an application is operating in a dynamic environment (e.g. an embedded system) it should be considered the software approach becomes slow. The speed at which the approach is able to evolve the efficient image operators automatically, determines a class of applications for which the approach is suitable for. The evolutionary image operator design has clearly demonstrated that very time consuming fitness calculation is the most painful problem of EHW. As soon as we are able to perform (at least) fitness calculation in hardware, we are also able to reduce design time substantially and potentially to support the continuous adaptation.

The idea of virtual RC initially appeared in [8], and was developed in the paper [11]. Some other authors have applied similar methods: Slorach and Sharman designed a simple fine-grained programmable logic array on top of an FPGA [12] and Torresen’s group approached a problem of context switching in this way too [13]. Macias has also implemented an 8 × 8 Cell Matrix using Xilinx Spartan-2 XC2S-200 FPGA. Other types of specialized platforms have been developed [14,15]. Note that none of these approaches deal with real-world applications of EHW operating in time-varying environments.

3.1 Description of the Virtual RC

The CGP instance (introduced in Section 2) is in fact implemented in an FPGA. As Fig. 2 shows the RC under design operates with nine 8 bit inputs (I0 – I8) and a single 8 bit output. It consists of 29 CFBs allocated in a grid of 8 columns and 4 rows. The last column contains only a single CFB, which is always connected to circuit output (a connection of the circuit output is not evolved). Every CFB has two inputs and can be configured to perform one of 16 functions listed in Fig. 3. Examples of a hardware implementation of some typical functions are obvious from the same figure.

Every CFB is also equipped with a register to support pipeline processing. A column of CFBs is considered as a single stage of the pipeline. In order to

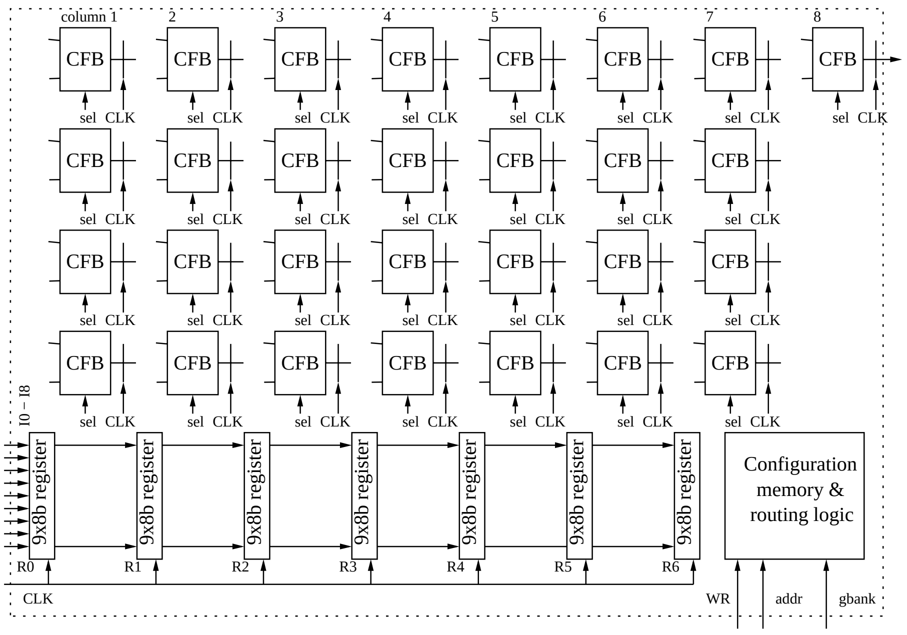

Fig. 2. A reconfigurable device for the evolutionary design of image operators.

synchronize CFBs with the processed inputs, we have to insert 72 bit (9 × 8 bits) registers R0 – R6 between neighboring columns. The CFBs' inputs are not connected to circuit inputs directly, but they are connected to the appropriate registers. It is more economical in hardware to allow a CFB's input is connected to one of 8 data sources than to one of 9 data sources because then only 3 address bits are needed for the routing multiplexer. Hence the following restrictions of conventional CGP (which are taken in account simultaneously) were introduced for this application to reduce the number of configuration bits:

– An input of each CFB placed in an odd row may be only connected to circuit's inputs I0 – I7 (via registers R0 – R6).
– An input of each CFB placed in an even row may be only connected to circuit's inputs I1 – I8 (via registers R0 – R6).
– An input of each CFB placed in the first (the second) column may be only connected to circuit's inputs via register R0 (R1, respectively).
– An input of each CFB placed in column 3 (4, 5, 6 or 7) may be connected to circuit's inputs via registers R1 and R2 (R2 and R3, R3 and R4, R4 and R5, R5 and R6, respectively) or to the output of a CFB, which is placed in some of the previous two columns.
– An input of the last CFB may be only connected to the output of a CFB, which is placed in the previous two columns.

Now we can derive the theoretical length of the configuration bitstream. Four bits are needed to determine the function in each CFB. Four bits also encode

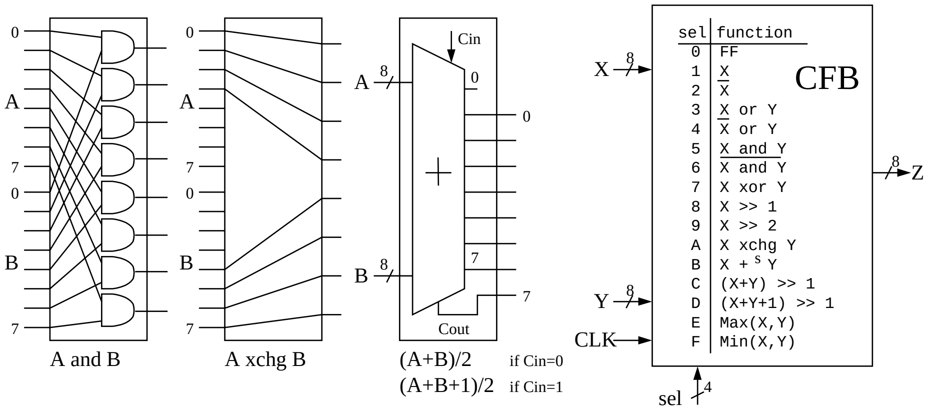

Fig. 3. An implementation of functions 5 (sel=0101), A (1010), C (1100), and D (1101) that are provided together with other functions in each CFB. Note that >> is a shifter and $+^s$ is an adder with saturation.

connection of every CFB's input in columns 3 – 7. We will utilize only three bits to do the same with inputs in columns 1, 2, and 8. In total, the length is: $9(3 + 3 + 4) + 20(4 + 4 + 4) = 330$ bits.

3.2 Routing Logic and Configuration Memory

The routing circuits are constructed using multiplexers that are controlled via bits of the configuration memory. Sixteen 16-input multiplexers are needed for selection of CFB's inputs. These inputs are usually taken either from the outputs of the eight CFBs in the previous two columns or from the previous two registers. Only 8-input multiplexers are required for CFBs of the first, the second, and the last columns since these CFBs utilize only eight possible inputs.

The configuration memory is implemented using flip-flops that are available in CLBs. Thus a two output CLB is employed as a two bit memory. Then all outputs of CLBs utilized for the configuration memory are connected to multiplexers that control the routing of CFBs. The configuration memory is depicted in Fig. 4. Four CFBs of a single column are configured simultaneously and their configuration is stored in a *cnfBank* configuration register. A column is selected via *addr* and a new configuration is uploaded using *gbank* configuration port and *WR* signal. Although the number of configuration bits needed for columns 1, 2 and 8 is lower than 48 bits, the configuration memory employs 48 bits per column in our prototype (in order to make the design easier). Hence the configuration memory consists of $48 \times 8 = 384$ bits.

The proposed configuration approach is a kind of partial reconfiguration that may be performed either form outside (if configuration signals are connected to circuit's pins) or from some internal circuits. Obviously, it is impossible to destroy the chip when a randomly generated configuration is uploaded.

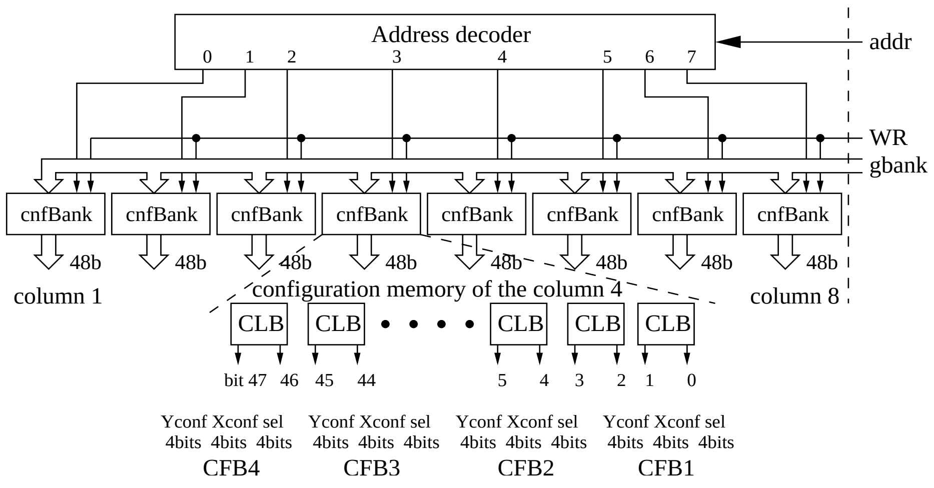

Fig. 4. The configuration memory is divided into eight configuration banks *cnfBank*. Each of them is responsible for configuration of a single column of CFBs.

3.3 Configuration Options

The virtual RC was described using VHDL at structural level but some elementary circuits (such as "Max" circuit) were defined behaviorally. Its functionality was tested in the *Active HDL v.3.6* environment.

Assume that the RC has been already configured and $L = 1$. Then the inputs (I0 – I8) are processed via the array of CFBs, and simultaneously, the inputs go through the register array R0 – R6. The first valid output is available in 9 clocks (a delay of R0 plus 8 CFBs), and the other ones each per a clock. Hence $K \times K$ pixels can be processed in $9 + (K \times K - 1)$ clocks. Maximal operational frequency is determined by a delay of a CFB and a delay of routing circuits (which is a single multiplexer).

It takes 8 clocks to configure the entire RC at the beginning of the computation. However, the reconfiguration process can be hidden totally. We can configure the first stage in the same time in which the first input values are stored into R0. Then every next stage can be configured while the previous stage processes data. Hence such a pipelined approach requires only 1 clock to reconfigure the entire device as seen in Fig. 5.

The RC in fact supports $L = 1$ as well as $L = 2$ setting. However if $L = 2$ is considered, every pixel must be sent to the RC two times to ensure synchronization of CFBs. Note that CFB's inputs may be connected to registers in two different stages (i.e. to values registered at two different moments) but a CFB must operate with the inputs registered at the same moment to produce a correct output. This leads to decreasing of the maximal frequency to a half if $L = 2$.

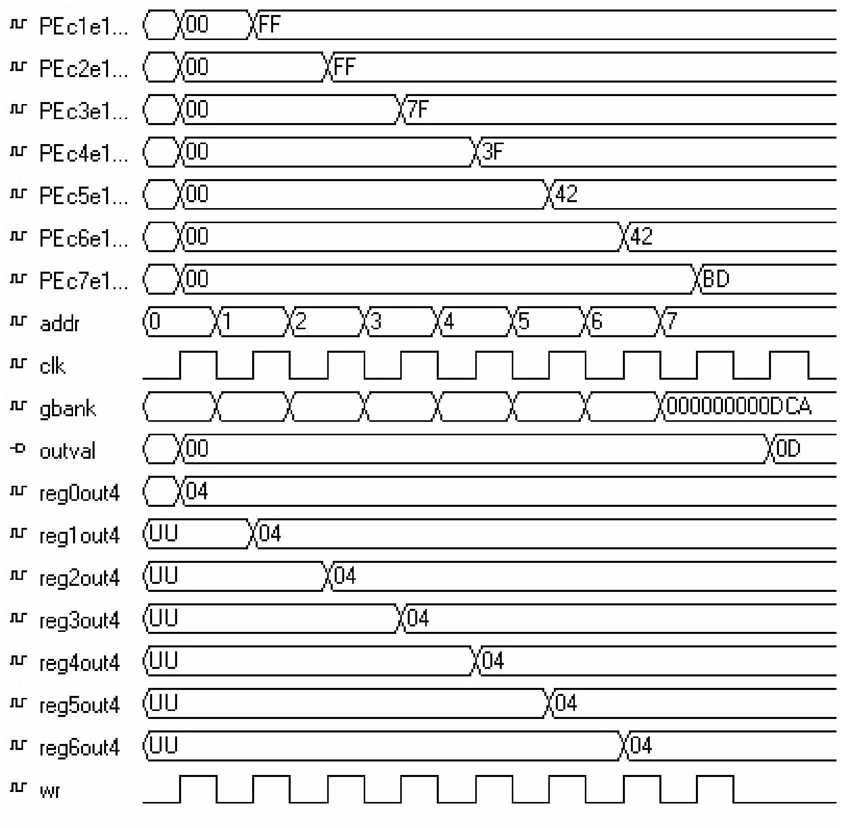

Fig. 5. An example of simultaneous pipelined reconfiguration and execution of the RC under design. We can observe the outputs of CFBs placed in the first row (*PEc*e1*) and the constant input I4 = 4, which goes via the register array (*reg*out4*). In the first clock, I4 is stored into *reg0out4*, and the first column of CFBs is configured using *gbank*, *addr* and *wr*. In the second clock, *PEc1e1* is calculated, I4 is stored into *reg1out4*, and the second column of CFBs is configured. The entire RC is configured in 8 clocks and the first output (*outval* = 0D) is available in the ninth clock.

3.4 Implementation Costs

In order to find out whether the design can be physically realized, we have tried to synthesize the RC into various FPGAs using *Xilinx Integrated Software Environment*. The results summarized in Table 1 qualify us to claim that it is possible to construct virtual reconfigurable devices for real-world applications. Furthermore it seems that genetic operators and fitness evaluation can be placed on the same FPGA (e.g. 77% XCV2000E's resources remain unused) and thus the entire application could be implemented in a single chip.

While we reached a sufficient operational frequency (at least for $L = 1$), higher implementation costs are main problem of the approach. If a fixed (either

Table 1. The results of the synthesis into various Xilinx Virtex FPGAs. Note that no timing constrains were defined for the synthesis program. The maximal operational frequency f_m is valid only for $L = 1$.

Family	Chip	Optimized for	Slices	Slices [%]	Equivalent gates	f_m [MHz]
Virtex E	XCV2000E	speed	4424	23	67937	96.4
Virtex 2	XC2V1000	speed	4879	95	74357	134.8
Virtex	XCV1000	speed	4489	36	68744	86.7
Virtex	XCV1000	area	4085	33	63773	81.3

conventional or evolved) operator is considered, about 2000 equivalent gates are needed for its implementation. The virtual RC (which can be configured to implement the same operator) requires 30 times more equivalent gates. A single CFB costs 1009 equivalent gates. So far our design consists of 29 CFBs, 55% gates needed to realize the entire virtual RC are devoted for implementation of routing multiplexers, configuration memory and the register array R0 – R6.

3.5 Fitness Calculation in Hardware

If $L = 1$ then the (virtual) RC can operate at $f_m = 134.8$ MHz. Thanks to pipelining, the RC is able to process each pixel at 134.8 MHz too. But furthermore, the *entire* evolvable system can operate at this frequency. Recall that a deterministic selection mechanism is applied in the evolutionary algorithm. Hence a new candidate circuit can always be generated simultaneously with fitness calculation of another circuit. As soon as the fitness calculation takes a half millisecond (e.g. it is 0.48 ms for 254×254 pixels), there is plenty of time to generate a new candidate circuit if the genetic unit takes place in the same FPGA. Because the reconfiguration time of the virtual RC is practically zero, the time of evolution is determined as

$$t_e = \frac{(K-2)^2.\mu.\rho}{f_m}$$

where $(K-2)^2$ is the number of processed pixels, μ denotes population size, and ρ is the number of generations.

Fig. 6 shows that pipelining of the RC can be extended to pipelining of the entire evolvable system. Every output value of the RC is compared with the corresponding pixel value of the original image. Then an absolute value of the difference is added to the content of an accumulator (which corresponds to fitness) in next clock. The circuit, which selects neighboring values of each pixel, is taken from [16]. FIFO is a K-pixel data structure that may be easily implemented in an FPGA. The original image has to be synchronized in similar way as seen in Fig. 6.

We can estimate how much time is needed to evolve a single image operator. Assume the setting $K = 256, \mu = 16, \rho = 160000$, which requires 1 day on PentiumIII/800 MHz in average. If $f_m = 134.8$ MHz, then a filter can be designed

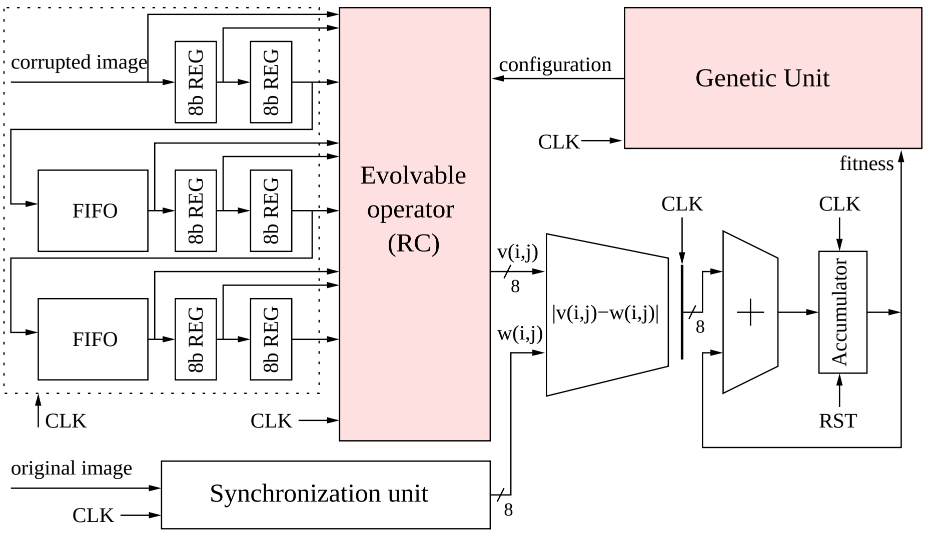

Fig. 6. Pipelining of the RC can be extended to the fitness calculation. FIFO ensures that each output pixel value is calculated using the right input values.

in $t_e = 20.4$ minutes in average using virtual RC. We obtained a speeding up more than 70. Some operators were evolved with $\rho = 20000$. If such a system is implemented in an FPGA, the adaptation time would be about $t_e = 2.6$ minutes. Assume that only $\frac{(K-2)^2}{2}$ pixels and $\rho = 10000$ are sufficient. Then we obtain the adaptation time $t_e = 38.3$ seconds which can be considered as a real-time adaptation in some applications. And finally, we can expect that it will be possible to implement in the future 16 virtual RCs in a single FPGA. Then all circuits of the population could be evaluated at the same time. Note that the fitness accumulator will have to be replicated 16 times, however only a single implementation of input FIFOs and registers is necessary. Such a system would be able to adapt to changing fitness function in 2.4 seconds.

4 Discussion

We did not implement the virtual RC in a physical FPGA yet. However, there is a company, which is interested to do it. Nevertheless the presented results show that an adaptive system for image pre-processing can be implemented in a common FPGA. If the approach is used only to speed up the evolutionary image operator design, then it should take several minutes to find a sufficient operator. If generality of the evolved operators is not required, it is possible to reduce the design time to tens of seconds. And that could be attractive for some real-time applications. Let us summarize the advantages of the proposed method for design of virtual reconfigurable devices:

– An architecture, which exactly fits to application requirements, can be developed for a given problem. It is mainly important for EHW, since hardware can reflect the representation used in the evolutionary algorithm.
– A style and granularity of reconfiguration may be defined according to the need of a given application.
– A virtual RC is described in a platform independent way. Hence it can be synthesized into various target devices and with various constraints.
– The process of a virtual RC design (i.e. its VHDL description) can be fully automatic. One can imagine a design environment, in which a designer is to press a button and a new virtual RC will be generated automatically according to the best parameters of RC and evolutionary algorithm found so far for a given EHW-based application.
– A description of a virtual RC can be offered as an IP macro.
– It is possible to implement very efficient context switching in Virtex FPGAs as it is evident in [13].

Higher implementation costs are the main disadvantages of the proposed method. Most resources available on an FPGA are wasted on configuration memory and routing logic implementation. It seems that the approach is suitable only for coarse-grained architectures since the scaling problem is not so painful. The scalability problem can be approached by reduction of configuration options (e.g. only local connection could be enabled only). The number of functions supported in a programmable element is not critical from a scalability viewpoint.

5 Conclusions

We have clarified that it is possible in principle to implement real-world applications of EHW in common FPGAs. We are going to realize the proposed RC in Virtex FPGA and to apply the method for design of other applications of EHW.

We have introduced a new level of abstraction to the hardware design. Virtual RCs are supposed to be beneficial not only for EHW, but also for the other research domains, e.g. reconfigurable computing.

Acknowledgment

The research was performed with the financial support of the Grant Agency of the Czech Republic under No. 102/01/1531 and 102/03/P004.

References

1. Yao, X., Higuchi, T.: Promises and Challenges of Evolvable Hardware. In: Proc. of the 1st International Conference on Evolvable Systems: From Biology to Hardware ICES'96, LNCS 1259, Springer-Verlag, Berlin (1997) 55–78
2. Higuchi, T. et al.: Real-World Applications of Analog and Digital Evolvable Hardware. IEEE Transactions on Evolutionary Computation, Vol. 3(3), (1999) 220–235

3. Vassilev, V., Miller, J.: Scalability Problems of Digital Circuit Evolution. In: Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society, Los Alamitos (2000) 55–64
4. Murakawa, M. et al.: Evolvable Hardware at Function Level. In: Proc. of the Parallel Problem Solving from Nature PPSN IV, LNCS 1141, Springer-Verlag Berlin (1996) 62–72
5. Hollingworth, G., Smith, S., Tyrrell, A.: 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, LNCS 1801, Springer-Verlag, Berlin (2000) 72–79
6. Miller, J., Job, D., Vassilev, V.: Principles in the Evolutionary Design of Digital Circuits – Part I. Genetic Programming and Evolvable Machines, Vol. 1(1), Kluwer Academic Publisher (2000) 8–35
7. Sekanina, L.: Image Filter Design with Evolvable Hardware. In: Applications of Evolutionary Computing – Proc. of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02, LNCS 2279 Springer-Verlag, Berlin (2002) 255–266
8. Sekanina, L.: Modeling of Evolvable Hardware. MSc. Thesis, Brno University of Technology (1999) 62 pp.
9. Sekanina, L., Drábek, V.: Automatic Design of Image Operators Using Evolvable Hardware. In: Proc. of the 5th IEEE Design and Diagnostic of Electronic Circuits and Systems DDECS'02, Brno, Czech Republic (2002) 132–139
10. Sekanina, L.: Evolution of Digital Circuits Operating as Image Filters in Dynamically Changing Environment. In: Proc. of the 8th International Conference on Soft Computing Mendel'02, Brno, Czech Republic (2002) 33–38
11. Sekanina, L., Růžička, R.: Design of the Special Fast Reconfigurable Chip Using Common FPGA. In: Proc. of the 3rd IEEE Design and Diagnostic of Electronic Circuits and Systems DDECS'00, Polygrafia Bratislava, Slovakia (2000) 161–168
12. Sloarch, C., Sharman, K.: The Design and Implementation of Custom Architectures for Evolvable Hardware Using Off-the-Shelf Progarmmable Devices. In: Proc. of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00, LNCS 1801, Springer-Verlag, Berlin (2000) 197–207
13. Torresen, J., Vinger, K. A.: High Performance Computing by Context Switching Reconfigurable Logic. In: Proc. of the 16th European Simulation Multiconference ESM 2002, Darmstadt, Germany (2002) 207–210
14. Haddow, P., Tufte, G.: Bridging the Genotype-Phenotype Mapping for Digital FPGAs. In: Proc of the 3rd NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society, Los Alamitos (2001) 109–115
15. Porter, R., McCabe, K., Bergmann, N.: An Application Approach to Evolvable Hardware. In: Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware, IEEE Computer Society (1999)
16. Fučík, O.: Reconfigurable Embedded Systems. PhD Thesis, Brno University of Technology (1997) 65 pp.