Towards Evolvable IP Cores for FPGAs

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

Abstract

The paper deals with a new approach to the design of adaptive hardware using common Field Programmable Gate Arrays (FPGA). The ultimate aim is to develop evolvable IP (Intellectual Property) cores. The cores should be reused in the same way as ordinary IP cores are reused. In contrast to the conventional cores, the evolvable cores are able to perform autonomous evolution of their internal circuits. The cores should be available in the form of HDL source code, i.e. they should be synthesizable into any reconfigurable device of a sufficient capacity. The approach is based on implementation of a virtual reconfigurable circuit and a genetic unit in an ordinary FPGA. In the presented case study an adaptive image filter is designed, implemented and synthesized. The proposed idea of evolvable IP core could open the way towards defining a business model for evolvable hardware.

1. Introduction

We can observe that FPGAs have made amazing advances in performance and capacity. Moreover, the component approach plays the dominant role not only in the hardware market. Some real-world applications of evolvable hardware have appeared in recent years. We do believe that it is the right time for integration and exploitation of these observations together.

Stoica et al. mentioned three years ago that "the path leads to the IP level and evolvable hardware solutions will become an integrated component in a variety of systems that will thus have an evolvable feature" [23]. No other (implementation) details were given there. However, there is a way how to implement such a component on FPGA. Hence we have developed the idea of *evolvable IP core* for FPGAs. The idea can be realized right now, however, it could be a promising approach especially for future days.

1.1. Component Technology

Nowadays software as well as hardware industry mainly benefits from *component* technology. Applications are composed of components that provide required behaviors via their interfaces. As examples we can mention TWAINPro component (allowing to acquire images from scanner devices) in the software domain [30] or processor Xilinx MicroBlaze in the hardware domain [31]. Designers have the access to various software and hardware components available in the market. They utilize the components as a "construction kit". The designer is to choose right components for a given application and to assemble the application from the components. Sometimes specialized components have to developed for a given application domain in order to meet uncommon constraints, such as the requirements for time, space, reliability, energy consumption or safety.

We can identify several reasons for the usage of the component approach. Primarily it is reduction of *time to market* (because of reusability) and increase in *reliability* (because target systems are built from verified and tested elements). Generally, the component approach leads to effective problem solving, especially in the case of complex systems. Furthermore, a reasonable *design methodology* can be established by means of reusable components. The component approach in fact determines the *business model* for producers.

In this paper we are interested in *reconfigurable hardware*, for instance, that devices based on FPGAs. The devices consist of a set of programmable blocks which provide some elementary functions and whose configurations and interconnection are defined by content of a *configuration memory*. It is usually possible to reprogram the chip (i.e. to modify some of the circuits realized in the device) even the chip performs computation.

Because the number of programmable blocks in FPGAs increases every year and the design methodology for designing complex systems is needed the component approach has been applied to the reconfigurable hardware in recent

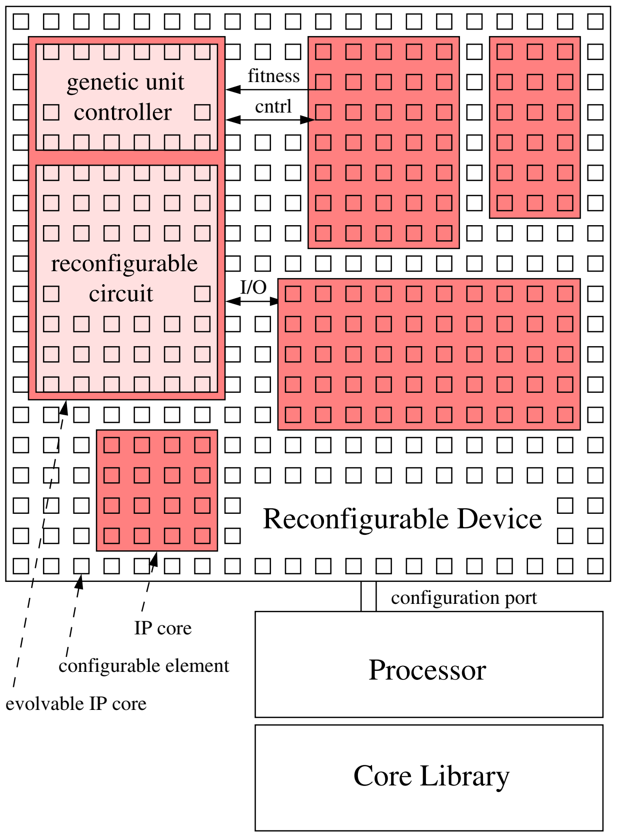


Figure 1. Common IP cores and an evolvable IP core in an FPGA

years.

As shown in Fig. 1 various circuits (components) are available in the *component library* as configuration bitstreams. Any such a circuit can be uploaded into the reconfigurable device on a position which has to be specified beforehand. Every circuit occupies some area in the reconfigurable device. The area is measured in programmable blocks or equivalent gates. A number of vendors offer the circuits that are usually referred to as IP cores to customers. The need for third party IP cores in FPGA designs is driven by increased size and performance of FPGA devices. Designers of these complex FPGA designs require proven IP cores as building blocks to accelerate systems development [10].

1.2. From Reconfigurable Computing and Evolvable Hardware to Evolvable IP Cores

Reconfigurable computing deals with *dynamic reconfiguration* of a device during the run [11, 2]. When several different IP cores are prepared beforehand then an ingenious mechanism can switch between them during execution in order to perform more computation in hardware than physical hardware resources enable. Higher performance is then achieved by (dynamically) building custom computational operators, pathways, and pipelines suited to specific properties of the task at hand. Designer has to divide the application to cores and to schedule the optimal sequence of configurations if timing is known beforehand. On the other hand when a request for a given type of operation (i.e. for an IP core which provides the operation) emerges during execution (the flow of program is not known in advance), online reconfiguration is done on the fly. Reconfigurable computing in fact provides a sort of *adaptability*—it is possible to change system function by means of reconfiguration if a requirement emerges. However, the repertoire of possible behaviors is not too wide. It is mainly determined by a set of components provided in the component library.

Adaptive hardware can be realized using *evolvable hardware* that combines reconfigurable devices with evolutionary algorithms [7, 16, 32]. Evolutionary algorithm is employed to assemble a target circuit according to the requirements formulated via fitness function. Furthermore, if the fitness function varies in time (i.e. the specification varies in time) and the application is suitable for evolvable hardware then the system should be able to adapt itself autonomously to a dynamic environment. New trends in evolvable hardware was summarized in [24]. It is evident from successful real-world applications of evolvable hardware that only a part of a system should be adaptive (evolvable). The other parts can be implemented by means of traditional “invariable” circuits. For instance, only pixel predictor is evolved in the image compression task; the remaining circuits (such as controllers, image reading and storage) are invariable [25]. It seems natural to reflect the situation within component context—some components should be *adaptive* and other should be *classical* (invariable).

In the context of components, we will call the evolvable part as evolvable component or evolvable IP core. Evolvable IP cores can be stored in the component library in the same way as ordinary IP cores are stored. However, after being uploaded and placed onto a reconfigurable device they will be able to evolve their internal circuits autonomously. When the adaptive behavior is not required then the evolvable IP core can be removed from the reconfigurable device.

Evolvable IP cores can be *reused* in the same way as ordinary IP cores can be. For instance, a vendor could sell an evolvable digital filter IP core instead of an “invariable” digital filter IP core. Evolvable IP cores could also open the way towards implementations of the adaptive reconfigurable computing and real-world applications of evolvable hardware in ordinary FPGAs—which is nowadays a problem to deal with. Let us note that the reusability has not been introduced for evolvable hardware yet.

1.3. Goals and Structure of the Paper

The objective of this study is to show how evolvable IP cores can be defined and realized using common reconfigurable platforms. In particular their internal architecture will be derived from the concept of *evolvable component*. The results we obtained during design, implementation, simulation and synthesis of a case application—adaptive image filter IP core—will be reported and analysed.

The rest of this paper is organized as follows. The general architecture of evolvable IP cores and requirements on evolvable IP cores will be introduced in Section 2. Section 3 describes an original approach to implementation of evolvable IP cores by means of ordinary FPGAs. In particular the concept of virtual reconfigurable circuit will be proposed. Section 4 discusses an example of evolvable IP core—adaptive image filter. The section deals with motivation, design, implementation and results of synthesis. Finally, conclusions and comments to the future research are given in Section 5.

2. Towards Evolvable IP Cores

2.1. System Decomposition

Evolvable IP cores are based on the theory of evolvable components that was introduced in [22] and developed in [18]. As shown in Fig. 1, the evolvable component consists of reconfigurable circuit, genetic unit and controller. (In this paper *reconfigurable circuit* denotes an internal part of the evolvable component; on the other hand *reconfigurable device* denotes the whole reconfigurable platform, for instance, an FPGA.) It is important that the genetic unit does not contain fitness calculation. It only implements genetic operators, chromosome memory and fitness memory. Fitness values are provided by an environment. The environment is to evaluate any configuration that was generated by the genetic unit and uploaded into the reconfigurable circuit. Evolvable component is in fact a generator of circuits that is controlled from the environment. The proposed approach represents a sort of system decomposition that enables us to reuse various parts of an originally coherent system.

Communication between the environment (that is represented by other components) and the evolvable component is as follows: First the component is initialized (initial population is generated). Then the following sequence of operations is repeated endlessly. The environment requires a new circuit to be uploaded into the reconfigurable circuit. The component is to generate a new configuration and to configure the reconfigurable circuit. If there is no configuration (chromosome) in the chromosome memory available then a new population has to be generated autonomously. When the evaluation of the circuit behavior is finished the fitness value is sent to the component. The component is to store the fitness value and to wait for another request from the environment. The environment can also require uploading of the best circuit that has been evolved so far. Hence the component has to continually store the best configuration that has been evolved so far and to provide it when requested. In all cases the evolvable component encapsulates the reconfiguration process that is *invisible* from the external environment.

It is supposed that a specialized evolvable IP core is developed for any specialized problem domain because genetic unit and the architecture of the reconfigurable circuit have to reflect the problem domain in order to outperform a random search. Because the fitness calculation is carried out outside the component, the component in principle supports dynamic fitness functions and open-ended evolution. The evolvable component can be implemented in software, as a single chip or as an evolvable IP core.

2.2. Implementation Issues

IP cores are available in the market as soft cores, firm cores and hard cores. We are interested in *soft cores* in this paper. It means that our objective is to design and implement evolvable IP cores at the level of HDL (Hardware Description Language) source code (e.g. in VHDL [1]). The advantage of this approach is that these evolvable IP cores will be represented in a platform independent format and thus they will work at various target architectures. However, any evolvable core must be able to reconfigure its internal reconfigurable circuit by *itself* (because it is evolvable)—and it is the main problem.

Let us consider the situation introduced in the previous paragraph more carefully. An evolvable IP core is downloaded from a component library and it has to be placed at a given position in the reconfigurable device. Then its internal reconfigurable circuit has to be reconfigured. It means that any internal programmable block of the reconfigurable device must be able to configure any other internal programmable block of the reconfigurable device because the evolvable IP core (its genetic unit) can be located anywhere in the array of programmable blocks. We assume that the reconfigurable device is homogenous. In other words, the *internal reconfiguration* must be supported in a given reconfigurable device (platform). However, common platforms (like FPGAs) do not support the internal reconfiguration—only the external reconfiguration is permitted via a specialized configuration interface (see Fig. 2).

According to our knowledge there exists only one platform that supports internal reconfiguration and all the features we require—Cell Matrix [12]. Except to Cell Matrix simulator, only a small chip containing several Cell Matrix cells has been developed. In order to accomplish our objectives and to utilize an evolvable IP core in a common FPGA and in a real-world application we have to employ another approach. Hence internal reconfigurable circuit in the evolvable IP core will be realized by means of the technique referred to as *virtual reconfigurable circuits* [20]. Let us note that the hardware virtualization is a common design technique for FPGA-based systems [4, 17]. We have extended the concept for designing virtual reconfigurable circuits.

3. Evolvable IP Cores in Ordinary FPGAs

3.1. Ordinary FPGAs

Some ordinary FPGAs support the *partial reconfiguration* which allows us to reconfigure only a portion of the array of programmable blocks while the remaining blocks can operate without being affected. For instance, the chips of Xilinx Virtex family can be partially reconfigured using JBits interface [8]. However, the format of configuration bitstream is not available to the public, the approach is not straightforward, and it is difficult to use it for our purposes. Atmel AT6000 is a SRAM-based device with dynamic partial reconfiguration and features very similar to Xilinx XC6200 family [28]. Under some conditions, it is possible to obtain description of the format of configuration bitstream and thus to build dynamically reconfigurable devices. Full reconfiguration takes about a millisecond; partial configuration is even faster. These Atmel FPGAs are relatively small devices if being compared to, as for instance, Virtex chips. All the devices are reconfigured via external pins.

3.2. Virtual Reconfigurable Circuit

Assume that our reconfigurable device (platform) is realized using Xilinx Virtex FPGA. IP cores are dynamically uploaded and removed into/from the FPGA. All the operations are performed by means of Virtex configuration port and JBits. When an evolvable IP core is uploaded then its configuration bitstream has to cause that there will be built following units at the specified position: virtual reconfigurable circuit, genetic unit, and controller. Fig. 2 shows that the virtual reconfigurable circuit is in fact a new reconfigurable circuit (consisting of eight programmable elements in our case) realized on top of an ordinary FPGA (using Virtex slices in our case). Virtex slices have to implement a new array of programmable elements, new routing circuits and new configuration memory. The virtual circuit can be configured internally or from FPGA's I/O pins if new configuration memory is connected to them.

The main advantage of the proposed method is that the array, the routing circuits and the configuration memory

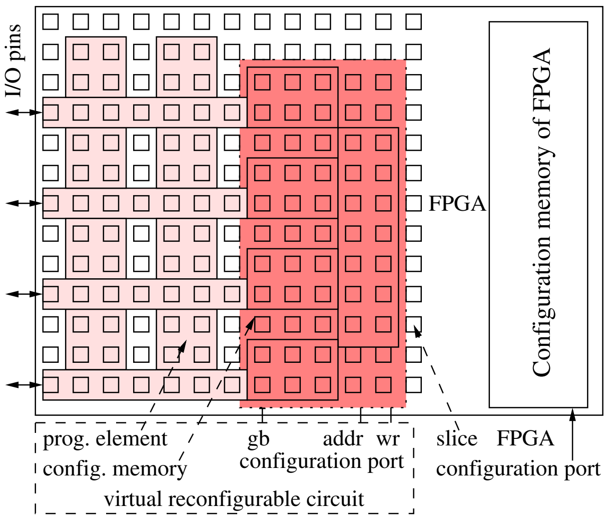


Figure 2. Virtual reconfigurable circuit realized using Virtex slices

can be designed exactly according to the requirements of a given application. Furthermore, style of reconfiguration and granularity of new virtual reconfigurable circuit can exactly reflect the needs of a given application. For instance, a pipelined partial one-clock reconfiguration operating with coarse-grained programmable elements supporting k functions (that we identified as crucial for purpose of our application) can be designed if required. By means of virtual reconfigurable circuits it is easy to insert domain knowledge not only to the genetic unit but also to the architecture of the reconfigurable circuit and thus to benefit from accurate implementation of circuit software model. Note that software simulations are usually performed in order to estimate the potential speeding up we could obtain in case of hardware implementation of a given application.

As an example, Fig. 3 shows one of "virtual" programmable elements of a virtual reconfigurable circuit consisting of eight of these elements and utilizing four inputs and two outputs. We will call the elements as Configurable Functional Blocks (CFB). Two configuration bits determine CFB's function; other four bits define the places where its inputs are connected to. This architecture is very similar to the representation employed in Cartesian Genetic Programming (CGP) that has been developed for circuit evolution [14]. The routing circuits are implemented using multiplexers. The configuration memory is composed of Virtex slices—a slice containing two flip-flops is utilized to store two bits of the memory. All bits of the configuration memory are connected to multiplexers that control routing and selection of functions in CFBs (see Fig. 3).

The number of CFBs utilized in the virtual reconfigurable circuit depends on a given application. Virtual recon-

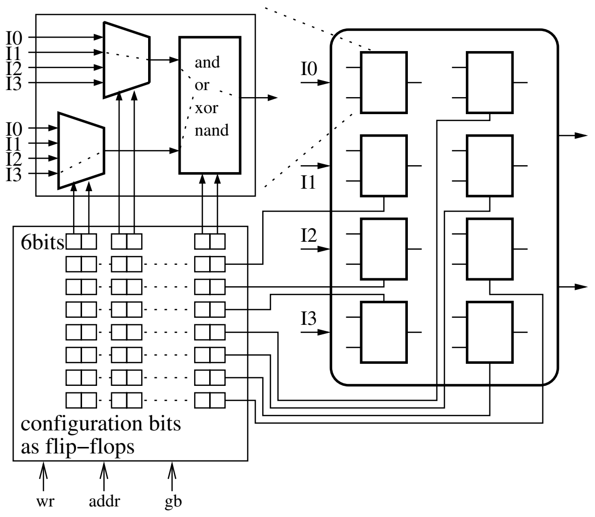


Figure 3. Implementation of a single CFB in a simple virtual reconfigurable circuit. The CFB is configured to realize (I0 or I3)

figurable circuits can be described in HDLs and synthesized with various constraints for various target platforms. Note that a partial reconfiguration is *not* required for a target platform at all. For instance, Xilinx XC4000 chips can be used for this purpose as well.

3.3. Genetic Unit and Controller

After a virtual reconfigurable circuit is prepared we have to implement a genetic unit and a controller. Generally, we have two options—either to employ a general processor or to design a specialized circuit. The specialized circuit is a sort of evolutionary algorithm implementation in hardware. A number of such implementations have been developed in evolvable hardware field, for instance see [9, 13, 26].

A number of soft IP cores that implement processors are available in the market or they are provided for free. For instance, Xilinx offers us MicroBlaze and PicoBlaze microcontroller IP cores [31] and Altera produces Nios core [27] etc. Another option is to utilize on-chip processors if they are available on a given target reconfigurable device. Xilinx Virtex II Pro XC2VP50 chip contains four PowerPC processors. The processor has to be programmed to execute the programme, which can communicate with the environment surrounding the core and to perform genetic operations over chromosomes. Furthermore, the processor is responsible for reconfiguration of the internal virtual reconfigurable circuit.

4. Case Study: Adaptive Image Filter as Evolvable IP Core

We have got very good experience with the evolutionary design of image operators. Hence we decided to realize an adaptive filter as an evolvable IP core. The research was motivated by real-world industrial applications in the domain of image recognition.

The objective is twofold. First, the proposed evolvable IP core should assist (or “replace”) the designer in the filter design phase. In this phase the designer has to find a structure and coefficients of the filter that suppresses a given type of noise. And it is a very time-consuming experimental job even for experts. Second, the evolvable IP core should operate in an image recognition FPGA-based embedded system. The core should be responsible for the real-time autonomous hardware adaptation to the changing type of noise. In the first case the proposed IP core should reduce the design time. In the second case an implementation in an FPGA-based embedded system (avoiding the usage of expensive personal computer) should exhibit high-performance computing at reasonable cost.

4.1. Software Simulations

We evolved a number of unique image operators and filters using a software simulator. The results reported in [19, 21] confirmed that the evolved filters are competitive with conventional filters in terms of quality and implementation costs. Furthermore, if more time is provided the evolution is able to adapt the filters to a time-varying type of noise (i.e. to a dynamic environment).

We also reported that the evolution can be accelerated more than 70 times if the fitness calculation is carried out in a physical circuit [20]. Then the adaptation time (which depends on the size of training images) should be sufficient for some real-time systems. In order to achieve this performance we had to implement a virtual reconfigurable circuit [20] supposing that the evolutionary algorithm is performed on a common personal computer. Note that conventional FPGAs such as Virtex devices cannot be applied directly because of problems with reconfiguration.

The following subsection describes how the virtual reconfigurable circuit is realized on top of a common Virtex FPGA. As far as our objective is to develop the adaptive filter as an evolvable IP core then the genetic unit and controller must be implemented together with the virtual reconfigurable circuit in hardware. Subsection 4.3 will introduce this implementation.

4.2. Virtual Reconfigurable Circuit for Evolution of Image Filters

The virtual reconfigurable circuit will be considered as a digital circuit of nine 8bit inputs I0 – I8 and a single 8bit output which processes gray-scaled (8bits/pixel) images. As seen in Fig. 4 every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbors of the processed image.

Fig. 4 shows that the circuit consists of 29 CFBs. Four CFBs are considered as a single stage of the pipeline. Each of CFBs has two 8bit inputs, a single 8bit output and can be configured to perform one of 16 relatively simple functions depicted in Fig. 5. The outputs of CFBs are equipped with registers. The circuit inputs are connected to CFBs via a set of registers R0 – R6. According to the configuration information, any CFB input can be connected either to one of the outputs of CFBs placed in the previous two columns or to one of circuit inputs (via appropriate register). It means that only combinational circuits with relatively short inter-block connections can be evolved.

The representation of a circuit in chromosome is very similar to the representation utilized in CGP at the functional level [15, 19]. It is necessary to operate at the functional level because of scaling problems. We have not been able to evolve an efficient image operator at the gate level yet. Note that functional level representation is usually referred to as coarse grained system in reconfigurable computing [6].

The configuration bitstream consists of three integers for each CFB. First two integers determine the places where CFB's inputs will be connected to. One of 16 functions is selected using the third integer. Typical length of configuration data of a single CFB is 12 bits. Note that the connection of circuit output is fixed (it is taken from the last CFB) and thus it is not evolved.

The configuration memory is implemented in the way depicted in Fig. 3. It is organized to 8 banks—each of them for a single stage (i.e. for four CFBs). In order to configure a single bank the required bank is selected using *addr*, 48 bit configuration is supplied to *gb* and *WR* signal is activated. While every bank contains 48 bits, $\frac{48}{2} \times 29 = 696$ Virtex slices are needed to implement the banks. The routing circuits are constructed by means of multiplexers that are controlled via bits of the configuration memory. Sixteen 16-input multiplexers are needed in order to select CFB's inputs.

The execution as well as the reconfiguration of the circuit is pipelined. The circuit can be configured in 8 clocks at the beginning of the computation. The first output value is valid by ninth clock (see the first arrow in Fig. 4). Circuit reconfiguration practically takes only a single extra clock because of pipelining as seen in Fig. 4 (second arrow).

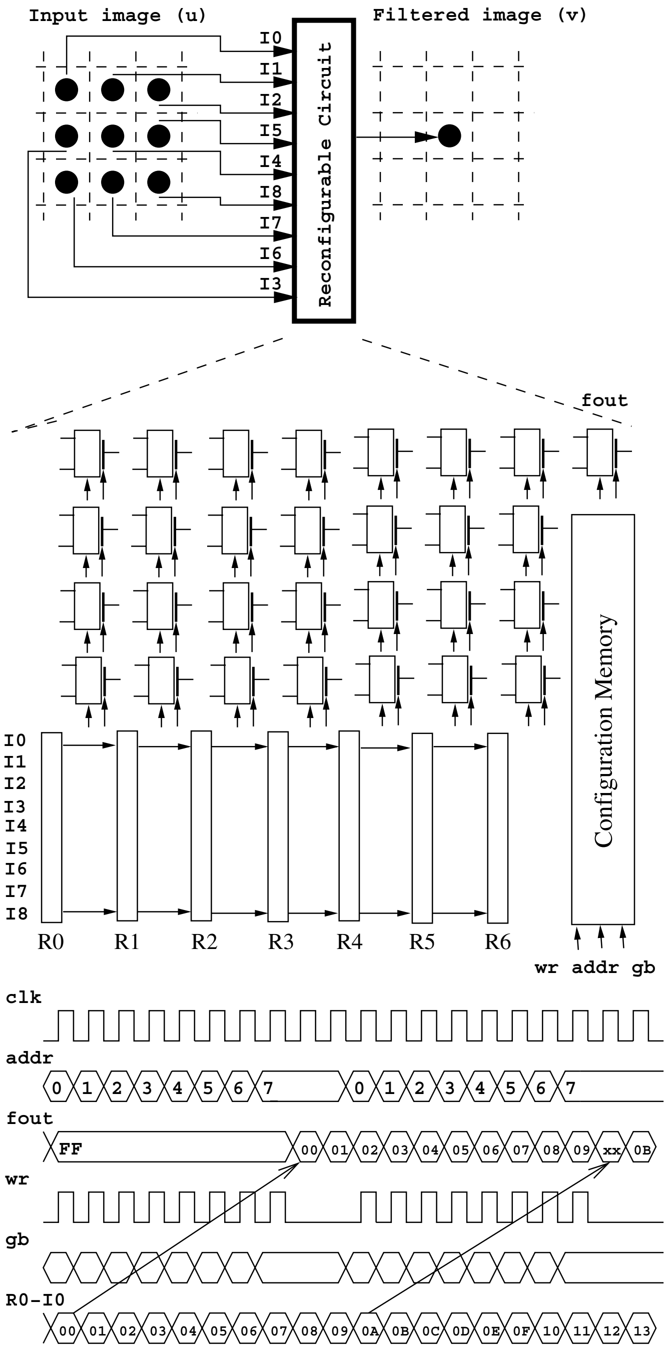
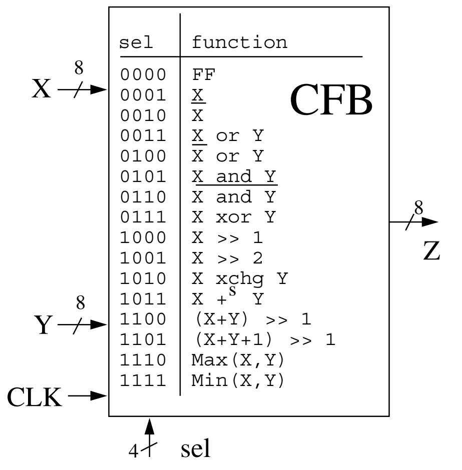


Figure 4. Virtual reconfigurable circuit for the evolutinary filter design. As example, the circuit is configured to send *I0* to *fout*. Its first configuration takes 8 clocks; the other reconfigurations take only one clock



sel	function
0000	FF
0001	$\underline{X}$
0010	X
0011	$\underline{X}$ or Y
0100	X or Y
0101	$\underline{\text{X and Y}}$
0110	X and Y
0111	X xor Y
1000	X >> 1
1001	X >> 2
1010	X xchg Y
1011	X $+^{S}$ Y
1100	(X+Y) >> 1
1101	(X+Y+1) >> 1
1110	Max(X,Y)
1111	Min(X,Y)

Figure 5. Any CFB supports 16 functions operating over 8 bits ($>>$ denotes a shifter and $+^{s}$ is adder with saturation)

In order to find out how many Virtex slices are needed for implementation of this virtual reconfigurable circuit we modeled the circuit by means of VHDL and synthesized the circuit into various FPGAs (XCV2000E, XC2V1000, and XCV1000). The best performance was obtained for XC2V1000 where the design costs 4879 slices (74357 equivalent gates) and it can operate at 134.8 MHz. For instance, the implementation using XCV1000 requires 4489 slices (68744 equivalent gates) and can operate at 86.7 MHz.

The maximum of operational frequency is valid only in the case that only neighboring columns in the array can be interconnected by evolution. Otherwise, every pixel must be sent to the circuit two times to ensure synchronization of CFBs. Note that CFB inputs could be connected to registers in two different stages (i.e. to the values registered at two different moments) but any CFB must operate with the inputs registered at the same moment to produce a correct output. This leads to decreasing of the maximum of frequency to a half.

For comparison, consider that a "fixed" conventional or evolved filter occupies approximately from 1000 to 5000 equivalent gates when implemented in the same FPGA. A single CFB occupies 1009 equivalent gates in the virtual reconfigurable circuit. Although the performance improvement is very positive (speeding up 70 times), implementation costs of the method are relatively high.

4.3. Implementation of Genetic Unit

Assume that the circuit operates at 134.8 MHz. Then a single image of 254 $\times$ 254 pixels can be filtered in 0.48 ms and the corresponding fitness value can be obtained at the same time [20]. Because of pipelining the reconfiguration can be performed in one clock (i.e. 7.4 ns). Hence the genetic unit must supply every new chromosome and so to reconfigure the virtual reconfigurable circuit every 0.48 ms.

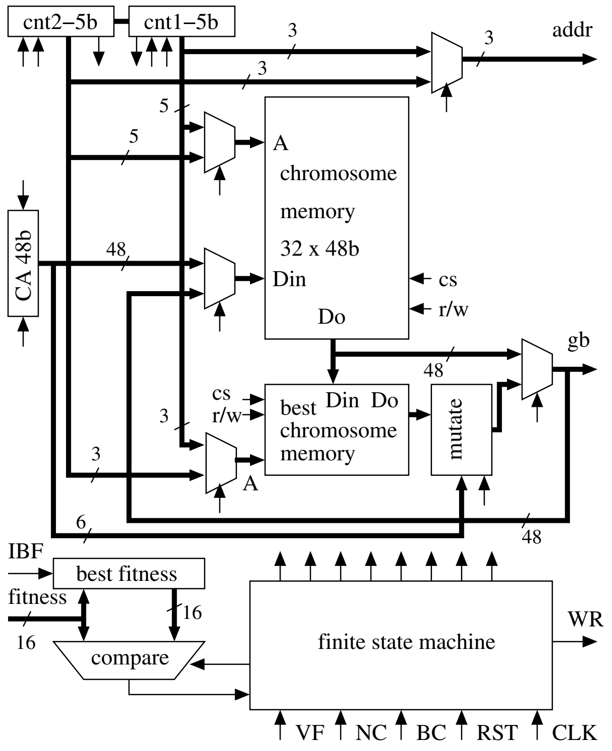


Figure 6. A simple genetic unit and controller

The genetic unit and the controller can be realized by means of a general processor, for instance, as Xilinx MicroBlaze IP soft core. It is a 32-bit RISC processor with Harvard architecture supporting code and data storage from either on-chip BlockRAM or off-chip RAM. When used with Virtex-II Pro, MicroBlaze occupies 900 Logic Cells and can operate at 150 MHz [31]. That is sufficient for the implementation of our genetic unit and controller. Furthermore, "a part of environment" of the evolvable IP core (e.g. fitness calculation) could also be represented (and performed) by the processor.

In order to compare various implementations we developed a simple genetic unit in VHDL. The unit is able to configure the virtual reconfigurable circuit and to communicate with the environment (see Fig. 6). The genetic unit implements a very simple variant of evolutionary algorithm utilizing only four individuals (very similar to the original evolutionary algorithm described in CGP [14]).

The unit consists of chromosome memory (containing four 8 $\times$ 48bit chromosomes), the best chromosome memory, the best fitness register, mutation unit, cellular automaton, two control counters and finite state machine that controls the unit and communicates with the environment. Because the virtual reconfigurable circuit has to be configured in 8 clocks per 48 bits (12 bits for each of four CFBs), the organization of chromosome memory is 32 $\times$ 48 bits and the best chromosome memory is 8 $\times$ 48 bits. When signal RST is activated, the chromosome memory is filled by data generated using a simple uniform cellular automaton operating according to rule 150. If the environment is willing to evaluate a configuration (NC=1) then a non-evaluated chromosome is sent to *gb* port (8 $\times$ 48 bits), addresses 0 –7 are generated at *addr* port and *wr* signal is activated to configure the virtual reconfigurable circuit.

After all chromosomes of the initial population had been evaluated and the environment had required another one, a mutated version of the best chromosome was sent to the configuration port. Only one randomly selected bit is inverted per 48 bits—its position is also determined by cellular automaton. If VF=1 then the fitness input holds a valid fitness value. This fitness value has to be compared with the best fitness value stored in the register. If the new fitness value is greater than the previous one, the previous value is replaced by the new one, and concurrently, the best chromosome memory is updated. Note that every new chromosome produced by the mutation unit is stored to the chromosome memory. The environment can set the best fitness value to zero using IBF signal. It can also request the unit to upload the best chromosome that has been evolved so far into the reconfigurable circuit (if BC=1). The controller is realized by means of the finite state machine that activates control signals of components depicted in Fig. 6.

The genetic unit was synthesized into various Virtex FPGAs. For instance, 2635 slices (38051 equivalent gates) are required for Virtex XC2V1000 and 2645 (40134 equivalent gates) for XCV1000.

4.4. Top Level

Fig. 7 shows the whole architecture of the evolvable IP core for adaptive image filtration. The environment is responsible for the fitness evaluation. It means that the environment (there is represented by conventional "invariable" circuits) has to provide training image as well as corrupted image and to add the differences of filtered image and training image in the fitness function (as it was explained in more details in [20]). An example of conventional components developed for image processing in FPGAs that could be utilized with the evolvable core is given for instance in [5]. Remember again that these circuits are not a part of the core. User is responsible for their implementation. On the other hand it gives user the opportunity to reuse the core in various applications.

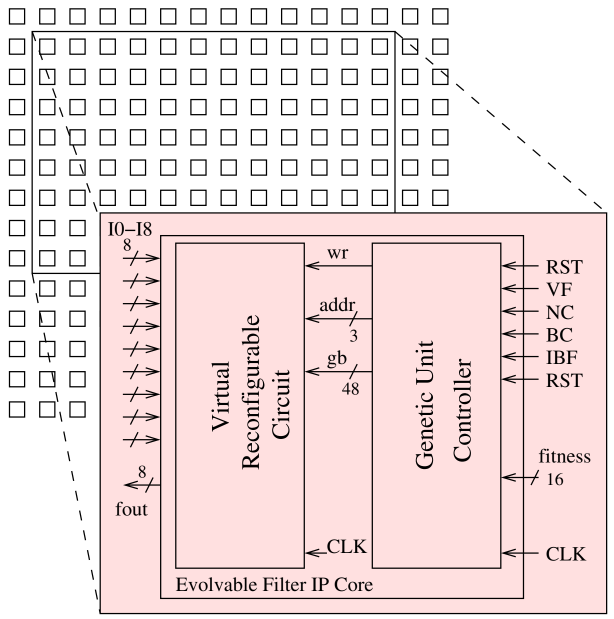


Figure 7. Evolvable filter as evolvable IP core

The whole evolvable IP core has not been synthesized yet. However, it is possible to estimate that it will cost about 100k equivalent gates if a "home-made" genetic unit is implemented as a specialized circuit according to Fig. 6. And such a core can definitely be realized on an ordinary FPGA.

4.5. Target Platform

Camea DX6 board is considered as a target reconfigurable platform [29]. The board has been developed for research purposes in the evolvable field; primarily focused on the use of dynamically reconfigurable hardware in real-world high-performance signal processing.

The DX6 system is a low cost, easy to integrate device capable of integrating embedded systems using a single PCB board. The DX6 consists of the following major components: DSP processor with VLIW architecture (C6000 family by Texas Instruments, Inc.), a Virtex FPGA, up to 128MB of operational memory, CPLD for inter-component communications and the FPGA configuration, a Flash ROM memory which stores both software (for DSP) and hardware (for FPGA) processing elements, six fast communication interfaces (more than 600Mb/s per channel) which can be used for general I/O interfacing and the Ethernet interface for an easy integration with desktop workstations.

5. Conclusions

We do believe that the proposed approach represents a step towards designing more adaptive and "soft" hardware.

It defines a new level of abstraction to the digital design. Furthermore, it introduces a sort of component approach to evolvable hardware. Evolvable IP cores are reusable and tunable easily since they are represented at the level of HDL source code. And these evolvable IP cores could in fact determine business model for evolvable hardware.

It was necessary to utilize a virtual reconfigurable circuit in order to implement the evolvable IP core. The application (designing of adaptive image filters) is a real-world problem and it is interesting for industry. Now we are able to solve the application using a relatively inexpensive ordinary FPGA. The proposed solution is expensive in terms of equivalent gates; however, it allows us speeding up the evolutionary design significantly.

We consider the proposed architecture and interface of the core as a reasonable minimum. For instance, we could enrich the core with a port for reading configurations which is necessary e.g. for image compression.

The proposed evolvable filter IP core can be utilized for the evolutionary design of image operators. Then it means that the result of the evolution has to be downloaded from the core and the evolved configuration bit stream can be reused as an "invariable" IP core in many applications. Because of hardware implementation we can reduce the design time (that is needed for evolution) from tens of hours to a few minutes or seconds. In this way more filters can be evolved and thus the designer could be "replaced" by a machine more effectively than by means of a software simulator.

However, evolvable IP cores are mainly devoted to implement adaptive and high performance real-time systems at reasonable cost. As examples of potential applications we can mention image compression [25] or dynamic hashing [3]. In these cases the evolvable core should contain two virtual reconfigurable circuits RC1 and RC2. RC1 implements the best circuit that has been evolved so far. Endless circuit evolution is performed in RC2. If a "better" circuit is evolved in RC2, its configuration is sent to RC1. Then RC1 represents the best response to the changing environment as a sequence of digital circuits.

The future work will be devoted to the synthesis of the whole evolvable image filter IP core, its implementation on DX6 board and its integration to an industrial image recognition system. Simultaneously we will develop CAD tools for the (semi)automatic design of evolvable IP cores.

6. Acknowledgments

The research was performed with the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based applications design methods* and the Research intention No. CEZ MSM 262200012 – *Research in information and control systems*.

References

[1] K. C. Chang. *Digital Systems Design with VHDL and Synthesis: An Integrated Approach*. IEEE Computer Society Press, 1999.

[2] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, 2002.

[3] E. Damiani, V. Liberali, and A. Tettamanzi. Dynamic Optimisation of Non-linear Feed-Forward Circuits. In J. Miller, A. Thompson, and T. C. Fogarty, editors, *Proc. of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00*, volume 1801 of *Lecture Notes in Computer Science*, pages 41–50, Edinburgh, Scotland, UK, 2000. Springer-Verlag.

[4] S. Dasasathyan, R. Radhakrishnan, and R. Vemuri. Framework for Synthesis of Virtual Pipelines. In *Proc. of the 15th International Conference on VLSI Design VLSID'02*, pages 369–374, Bangalore, India, 2002. IEEE Computer Society.

[5] O. Fučík. *Reconfigurable Embedded Systems*. PhD thesis, Brno University of Technology, Faculty of Electrical Engineering and Computer Science, 1997. (In Czech).

[6] R. Hartenstein, M. Herz, T. Hoffmann, and U. Nageldinger. Generation of Design Suggestions for Coarse-Grain Reconfigurable Architectures. In R. Hartenstein and H. Grünbacher, editors, *Proc. of the 10th International Conference on Field-Programmable Logic and Applications FPL2000*, volume 1896 of *Lecture Notes in Computer Science*, pages 389–399, Villach, Austria, 2000. Springer-Verlag.

[7] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya. Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In *Proc. of the 2nd International Conference on Simulated Adaptive Behaviour*, pages 417–424. MIT Press, 1993.

[8] JBits documentation, Xilinx, Inc., 1999.

[9] I. Kajitani, T. Hoshino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, M. Iawta, D. Keymeulen, and T. Higuchi. A Gate Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI. In M. Sipper, D. Mange, and A. Perez-Uribe, editors, *Proc. of the 2nd International Conference on Evolvable Systems: From Biology to Hardware ICES'98*, volume 1478 of *Lecture Notes in Computer Science*, pages 1–12, Lausanne, Switzerland, 1998. Springer-Verlag.

[10] M. Keating and P. Bricand. *Reuse Methodology Manual for Systems on a Chip Designs*. Kluwer Academic Publishers, 1999.

[11] F. Kurdahi, N. Bagherzadeh, P. Athanas, and J. Munoz. Configurable Computing. *IEEE Design and Test*, January-March 2000.

[12] N. Macias. The PIG Paradigm: The Design and Use of a Massively Parallel Fine Grained Self-Reconfigurable Infinitely Scalable Architecture. In A. Stoica, D. Keymeulen, and J. Lohn, editors, *Proc. of the 1st NASA/DoD Workshop on Evolvable Hardware*, pages 175–180, Pasadena, CA, USA, 1999. IEEE Computer Society.

[13] 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.

[14] 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.

[15] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi. Evolvable Hardware at Function Level. In *Parallel Problem Solving from Nature PPSN IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 62–71. Springer-Verlag, 1996.

[16] E. Sanchez and M. Tomassini. *Towards Evolvable Hardware: The Evolutionary Engineering Approach*, volume 1062 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[17] H. Schmit. Incremental Reconfiguration for Pipelined Applications. In *Proc. of the 5th Symposium on FPGA-Based Custom Computing Machines FCCM'97*, pages 47–55. IEEE Computer Society, 1997.

[18] L. Sekanina. *Component Approach to Evolvable Systems*. PhD thesis, Brno University of Technology, Faculty of Information Technology, 2002.

[19] L. Sekanina. 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*, volume 2279 of *Lecture Notes in Computer Science*, pages 255–266, Kinsale, Ireland, 2002. Springer-Verlag.

[20] L. Sekanina. Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. In A. Tyrrell, P. Haddow, and J. Torresen, editors, *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.

[21] L. Sekanina and V. Drábek. Automatic Design of Image Operators Using Evolvable Hardware. In *Proc. of the 5th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop DDECS2002*, pages 132–139, Brno, Czech Republic, 2002. Brno University of Technology.

[22] L. Sekanina and A. M. Sllame. Toward Uniform Approach to Design of Evolvable Hardware Based Systems. In R. Hartenstein and H. Grünbacher, editors, *Proc. of the 10th International Conference on Field-Programmable Logic and Applications FPL2000*, volume 1896 of *Lecture Notes in Computer Science*, pages 814–817, Villach, Austria, 2000. Springer-Verlag.

[23] 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.

[24] A. Stoica, D. Keymeulen, R. S. Zebulum, M. I. Ferguson, and X. Guo. On Two New Trends in Evolvable Hardware: Employment of HDL-based Structuring, and Design of Multi-functional Circuits. In *Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 56–59, Alexandria, USA, 2002. IEEE Computer Society.

[25] M. Tanaka, H. Sakanashi, M. Salami, M. Iwata, T. Kurita, and T. Higuchi. Data Compression for Digital Color Electrophotographic Printer with Evolvable Hardware. In M. Sipper, D. Mange, and A. Perez-Uribe, editors, *Proc. of the 2nd International Conference on Evolvable Systems: From Biology to Hardware ICES'98*, volume 1478 of *Lecture Notes in Computer Science*, pages 106–114, Lausanne, Switzerland, 1998. Springer-Verlag.

[26] 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.

[27] Altera home page, 2003. http://www.altera.com.

[28] Atmel home page, 2003. http://www.atmel.com.

[29] Camea home page, 2003. http://www.camea.cz.

[30] Component source homepage, 2003. http://www.componentsource.com.

[31] Xilinx microblaze ip core, 2003. http://www.xilinx.com.

[32] 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.