An evolvable hardware system in Xilinx Virtex II Pro FPGA

Zdeněk Vašíček and Lukáš Sekanina*

Faculty of Information Technology,
Brno University of Technology,
Bozetechova 2, Brno 612 66, Czech Republic
E-mail: vasicek@fit.vutbr.cz
E-mail: sekanina@fit.vutbr.cz
*Corresponding author

Abstract: In this paper, a new circuit architecture for image filter evolution is proposed. The evolvable system is based on the implementation of a search algorithm in the PowerPC processor which is available in Xilinx Virtex II Pro Field Programmable Gate Arrays (FPGAs). Candidate filters are evaluated in a domain-specific virtual reconfigurable circuit implemented using a reconfigurable logic of the same FPGA. As the PowerPC processor enables to execute more sophisticated search algorithms than an original solely circuit-based solution by Martinek and Sekanina, a higher performance can be obtained. In the FPGA, a resulting human-competitive filter can be evolved in 15 sec in average.

Keywords: field programmable gate array; FPGA; evolutionary algorithm; EA; evolvable hardware; image operator.

Reference to this paper should be made as follows: Vašíček, Z. and Sekanina, L. (2007) 'An evolvable hardware system in Xilinx Virtex II Pro FPGA', *Int. J. Innovative Computing and Applications*, Vol. 1, No. 1, pp.63–73.

Biographical notes: Zdeněk Vašíček received an MSc in the Electrical Engineering and Computer Science from the Faculty of Information Technology, Brno University of Technology, Czech Rep. in 2006. Currently, he is a PhD student at the Faculty of Information Technology. His research interests are focused on the area of evolvable hardware. He was awarded the J. Hlavka award in 2006.

Lukáš Sekanina received his PhD and all his degrees from Brno University of Technology, Czech Republic. Currently, he is an Associate Professor at the Faculty of Information Technology, Brno University of Technology. He was a Fulbright scholar with NASA Jet Propulsion Laboratory, Pasadena (2004), a Visiting Lecturer with Pennsylvania State University (2001) and a visiting researcher with Department of Informatics, University of Oslo, Norway (2001). He is author of monograph Evolvable Components (Springer Verlag, 2004) and (co)author of more than 50 refereed conference/journal papers mainly on evolvable hardware. His research interests are focused on the theory, design and hardware implementations of bio-inspired computing systems.

1 Introduction

The idea of evolutionary hardware design was introduced at the beginning of 1990s in papers Higuchi et al. (1993) and de Garis (1993). Evolvable hardware is usually defined as an approach in which the Evolutionary Algorithm (EA) is utilised to search for a suitable configuration of a reconfigurable device in order to achieve such the circuit behaviour which satisfies a given specification. Various reconfigurable devices have been utilised so far; however, Field Programmable Gate Arrays (FPGAs) remain the most popular digital reconfigurable devices in the evolvable hardware community.

With the development of a deep submicron semiconductor technology, FPGAs are becoming more complex, that is, containing more configurable elements and more configurable interconnects. In order to increase their performance, advanced non-configurable hard cores (such as Block RAMs (BRAMs) and multipliers) have also been integrated on a chip. The most advanced FPGAs integrate a reconfigurable logic, non-configurable hard cores as well as general-purpose processors on a single chip. Xilinx has introduced PowerPC processors in its Virtex II Pro family, see Xilinx Inc. (2005). Atmel has offered a Field-Programmable System Level Integrated Circuit (FPSLIC) chip which combines an AT40K FPGA with AVR microcontroller, see Atmel Corp. (2002).

In order to exploit these embedded processors for purposes of evolvable hardware, Glette and Torresen (2005) have utilised Xilinx Virtex II Pro FPGA chip to evolve small combinational circuits, such as 2-bit multiplying benchmark circuits. The contribution of their work is that they implemented the EA in the PowerPC processor while candidate circuits were evaluated in the array of programmable elements of the same chip. As the PowerPC processor can directly be connected via a fast local bus to programmable elements of the FPGA, the approach simultaneously benefits from a fast evaluation of candidate circuits directly in the FPGA and from a software implementation of EA which can be more sophisticated than a potential circuit implementation of the EA. Although only initial experimental results were reported, their work indicates that by using a better search algorithm in the PowerPC processor, it should be possible to improve the search process and so to reduce the evolution time.

Evolvable hardware is well suited for adaptive image processing systems, mainly because some intelligent preprocessing is usually required in these systems as the input data streams come from complex real-world situations via non-ideal cameras. Among these applications, image/video preprocessing (filtering), segmentation, recognition and compression can be included.

A number of papers have dealt with the evolutionary image filter design at the hardware level (see e.g. Burian and Takala, 2004; Dumoulin et al., 2000; Erba et al., 2001; Hollingworth et al., 1999; Porter, 2001; Sekanina, 2004; Sekanina and Ruzicka, 2003; Smith et al., 2005; Zhang et al., 2004a,b). In most cases, candidate circuits were evaluated in a reconfigurable device while EA was executed on a personal computer or a cluster of workstations. In our recent work, we have introduced a concept of the complete hardware implementation of the image filter evolution for FPGAs (see Martinek and Sekanina, 2005). By *complete hardware implementation* we mean that the evolving filter as well as the EA are implemented on a single chip as application-specific digital circuits. The primary advantages of this approach are high speed, low cost and potentially low power consumption in comparison with a solution which utilises a common PC.

This paper explores the idea of Glette and Torresen for a real-world application – image filter evolution. The objective of this work is to design, implement and evaluate a system for image filter evolution on a single Virtex II Pro FPGA chip. The solution is based on a software implementation of EA carried out in the PowerPC processor and on the utilisation of the reconfigurable array of the same FPGA for purposes of candidate filters evaluation. It is expected that the proposed implementation will be more efficient than a processor-less FPGA implementation of this task reported by Martinek and Sekanina (2005). These FPGA implementations will be compared on two design tasks:

1 the design of a shot noise removal filter and

2 the design of an edge detection operator.

2 Modern FPGAs

Figure 1 shows a typical architecture of a Xilinx FPGA, which is a two-dimensional array of reconfigurable resources that include the Configurable Logic Blocks (CLBs), Programmable Interconnect Blocks (PIB) and reconfigurable I/O Blocks (IOB). A CLB consists of four slices; each of them contains two function generators, two flip-flops and some additional logic. The FPGAs operate according to a configuration bitstream that is stored in the configuration SRAM memory. By writing to the configuration memory, the user can physically create new (digital) electronic circuits. The advantage of FPGAs is that a new hardware functionality is obtained through a simple reprogramming of the chip.

Figure 1 FPGA Virtex II Pro architecture which contains two PowerPC processors

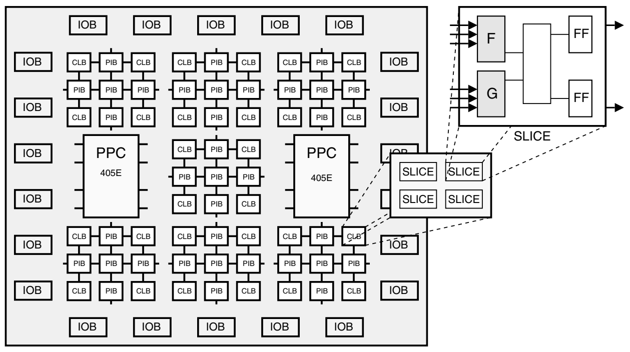


Figure 1 also demonstrates that FPGAs can contain embedded processor cores. Currently, two such families are available on the market: FPSLIC by Atmel and Virtex II Pro (resp. Virtex 4) by Xilinx. As this work is based on Xilinx FPGAs, we will describe only the Virtex II Pro family. The Virtex II Pro FPGAs provide up to two PowerPC405 processors, 32-bit RISC processor cores in a single device. These industry-standard processors offer high performance and a broad range of third-party support. The IBM PowerPC 405 core is integrated into the Virtex-II Pro device using the IP-Immersion architecture which allows hard IP cores to be diffused at any location deep inside the FPGA fabric. The processor core operates at a maximum frequency of 400 MHz. As shown in Figure 2, the PowerPC 405 processor contains the following elements:

- a five-stage pipeline consisting of fetch, decode, execute, write-back, and load writeback stages
- a virtual-memory-management unit that supports multiple page sizes and a variety of storage-protection attributes and access-control options
- separate 16 kB instruction-cache and data-cache units
- three programmable timers
- On-Chip Memory (OCM) controller and
- variety of interfaces, including: Processor Local Bus (PLB) interface, Device Control Register (DCR) interface, clock and power management interface and JTAG port interface.

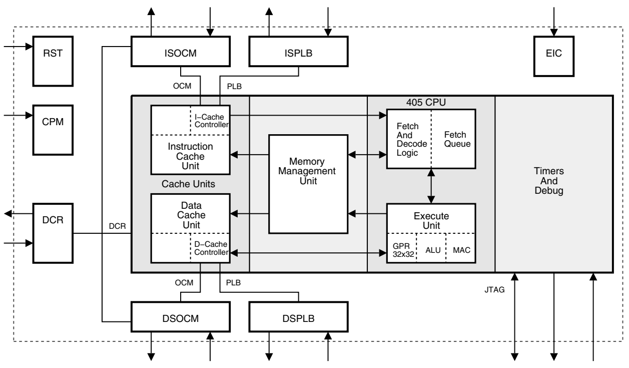
Table 1 summarises basic parameters of the PowerPC 405 interfaces. Although the PLB controller is more complicated than OCM controller, it provides a higher throughput. Further details of the PowerPC 405-processor architecture are available in Xilinx Inc. (2005).

Table 1 A comparison of basic interface parameters of PowerPC 405

Interface	*DCR*	*IS-OCM*	*DS-OCM*	*IS-PLB*	*DS-PLB*	*C405*
Throughput [MB/s]	300	1200	600	1200	1200	1200
Data bus [b]	32	64	32	64	64	32
Addr. space [B]	1 k	16 M	16 M	4 G	4 G	4 G
Variable latency	Yes	No	No	Yes	Yes	–

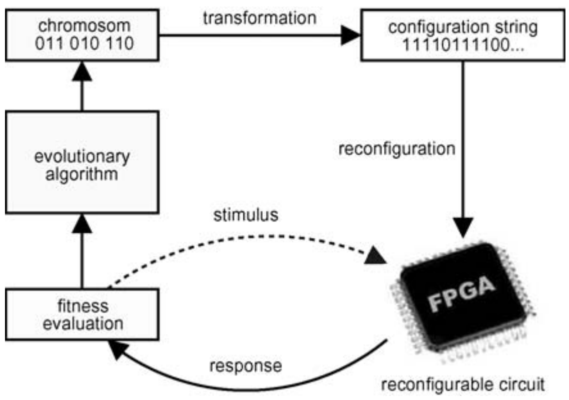
Note: IS: Instruction Side; DS: Data Side. The C405 column summarises the performance of the whole PowerPC core.

Figure 2 Architecture and interface of the PowerPC 405 processor



Source: see Xilinx Inc. (2005).

Figure 3 Evolvable hardware: candidate configurations are generated by the EA, uploaded to a reconfigurable device and evaluated using the fitness function



3 Evolvable hardware and FPGAs

3.1 Elementary principles of evolvable hardware

Evolvable hardware is an approach in which a physical hardware is created and adapted using the EA (Higuchi et al., 1993). Figure 3 shows that digital circuits are encoded as chromosomes of the EA. In order to evaluate a candidate circuit, a new configuration is created on the basis of the chromosome content. This configuration is uploaded into the FPGA and evaluated for a chosen set of input stimuli. For a particular task, the circuit quality is expressed by a fitness value – simply by a real or integer number. The fitness function (which calculates the fitness value) can include behavioural as well as non-behavioural requirements (e.g. reflecting thus functionality versus circuit size). After evaluation of all candidate circuits of a population, a new population of candidate circuits can be produced. That is typically ensured by applying genetic operators (such as mutation and crossover) on existing circuit configurations. High-scored candidate circuits have got a higher probability that their genetic material (configuration bitstreams) will be selected to next generations. The process of evolution is terminated when a perfect solution is obtained or when a certain number of generations are evaluated.

As the EA is a stochastic algorithm, the quality of resulting circuits is not guaranteed. However, the method has some advantages: Firstly, artificial evolution can sometimes produce intrinsic designs for electronic circuits which lie outside the scope of conventional methods (see Thompson et al., 1999). Secondly, the challenge of conventional design is replaced by that of designing an evolutionary process that automatically performs the design in our place. This may be harder than doing the design directly, but makes autonomy possible (see Stoica, 2004).

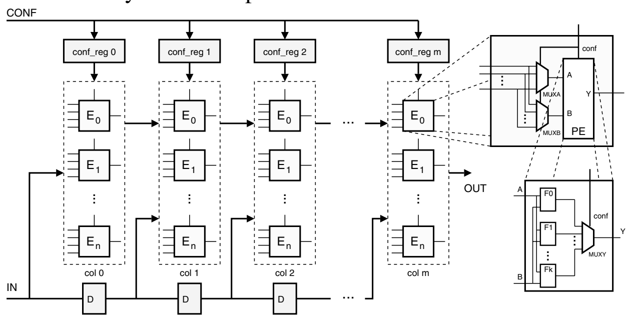
An efficient and fast reconfiguration subsystem is a desired feature for building evolvable hardware applications. Most FPGA families can be configured only *externally*. The *internal reconfiguration* means that a circuit placed *inside* the FPGA can configure the programmable elements of the same FPGA. Although the Internal Configuration Access Port (ICAP) has been integrated into the Xilinx Virtex II family (Blodget et al., 2003), it is still too slow for evolvable hardware applications. In order to overcome the problem of slow reconfiguration, Sekanina (2004) has developed Virtual Reconfigurable Circuits (VRCs). The use of VRCs has allowed us to introduce a novel approach to the design of complete evolvable systems in a single FPGA – Sekanina (2004), Sekanina and Friedl (2004) and Martinek and Sekanina (2005).

3.2 The concept of VRC

The VRC is, in fact, a second reconfiguration layer developed on the top of an FPGA in order to obtain a fast reconfiguration and application-specific programmable elements. Figure 4 shows that the VRC consists of an array of programmable elements E_i. Each of them can be connected to some programmable elements located in a previous column or to some of primary inputs. The routing circuits are created using multiplexers. Any programmable element can be configured to perform one of k functions. The configuration memory of the VRC is typically implemented as a register array. All bits of the configuration memory are connected to multiplexers that control the routing and selection of functions in programmable elements. The computation of VRC is pipelined since a single column of programmable elements acts as a single stage of the pipeline. The values coming from primary inputs are synchronised with the computation of programmable elements via a set of registers.

Because the array of programmable elements, routing circuits, configuration memory, style of reconfiguration and granularity of a VRC can be designed exactly according to the requirements of a given application, designers can create an optimised application-specific reconfigurable device. Furthermore, the VRC is described in a Hardware Description Language (HDL), that is, independently of a target platform. It is crucial from our perspective that the VRC can directly be connected to an EA implemented on the same chip. The EA can be implemented either as an application-specific circuit or as a program for an embedded processor. If the structure of the chromosome corresponds to the configuration interface of the VRC then a very fast reconfiguration can be achieved (e.g. consuming a few clock cycles only) – which is impossible by means of any other technique.

Figure 4 Virtual reconfigurable circuit: A second configuration layer on the top of an FPGA



3.3 FPGA implementations of evolvable systems

The FPGA-based implementations of evolvable hardware systems can be divided into two groups:

- The FPGA serves in the fitness calculation only. The EA (which is usually executed on a personal computer) sends configuration bitstreams representing candidate circuits to the FPGA in order to obtain their fitness values. The FPGA is configured via an external configuration port (i.e. via SelectMAP, JTAG or XHWIF interface which is provided with JBits in Xilinx FPGAs). Thompson, who has evolved an innovative implementation of a tone discriminator, pioneered this approach. He has employed Xilinx XC6216 FPGA and directly utilised configuration bitstreams of the FPGA as chromosomes. Note that it is practically impossible to perform the evolutionary design directly at the level of Virtex configuration bitstreams because the Virtex device can easily be damaged by uploading a randomly generated bitstream.
- The entire evolvable system is implemented in an FPGA. The idea of the complete hardware evolution for FPGAs was initially demonstrated by Tufte and Haddow (2000); however, they provided only a simple example of the optimisation of a few FIR filter coefficients stored in a register.

Table 2 surveys examples of FPGA implementations of digital evolvable systems. In each of these implementations, we can identify the following components: an array of reconfigurable elements, an EA, a fitness calculation unit and a controller. The problem domain determines the type of reconfigurable elements. In some cases the evolution is performed directly with reconfigurable cells of an FPGA; in other cases a kind VRC is utilised. An evolutionary optimisation of coefficients stored in registers represents the simplest example. The EA and fitness calculation unit can be implemented either as an application specific circuit or as a program. The program is running either in a personal computer or in an embedded processor which is integrated into the FPGA. The embedded processor is typically available as a hard core (e.g. PowerPC in Virtex II Pro FPGA) or as a soft core (a specialised configuration of an FPGA (e.g. the MicroBlaze core by Xilinx Inc. (2006)).

4 A new architecture for image filter evolution

The goal of this work is to find a structure and coefficients of a 3×3 image filter for a given type of noise. We will follow the approach introduced by Sekanina (2002). Later Zhang et al. (2004b) proposed a solution in which a reconfigurable filter is implemented as a VRC in the FPGA and EA is running on a personal computer. Martinek and Sekanina (2005) implemented a very similar architecture; however, their EA was created as an application specific digital circuit on the same FPGA as the VRC. The proposed solution utilises the PowerPC processor available in the Virtex II Pro FPGA to implement the EA.

4.1 Image filters

Every image operator will be considered as a digital circuit of nine 8-bit inputs and a single 8-bit output, which processes gray-scaled (8-bit/pixel) images. Conventional solutions typically utilise a convolution operator or some non-linear operators (see Sonka et al., 1999). In case of convolution filters, we are interested in the spatial domain where the input image convolves with the filter function f (see Figure 5). Then the task is to find the values of the so-called *convolution kernel*. In case of non-linear filters, a non-linear operator has to be developed. The median operator is the most known example of a non-linear filter. As there is not any suitable theory for the design of non-linear operators, evolutionary design techniques have been utilised to accomplish this task in the recent years.

Figure 5 A new pixel value calculation using a convolution kernel 3 × 3, $h[i, j] = f(p1, \ldots, p9)$

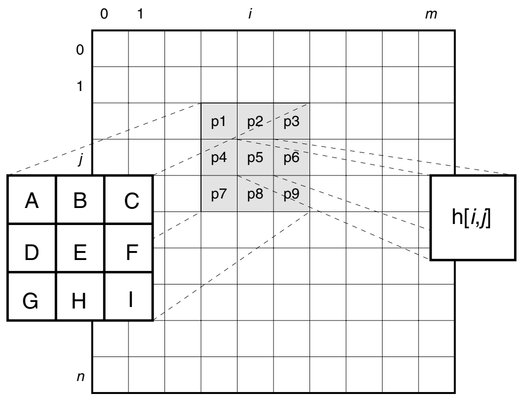


Table 2 Examples of FPGA implementations of evolvable digital systems

Reference	*Application*	*Platform*	*EA*	*Fitness*
External reconfiguration				
Thompson et al. (1999)	Tone discriminator	XC6216	PC	PC
Huelsbergen et al. (1999)	Oscillators	XC6216	PC	PC
Zhang et al. (2004b)	Image filters	VRC	PC	PC
Gordon (2005)	Arithmetic circuits	Virtex CLB	PC	PC
Gwaltney and Dutton (2005)	IIR filters	VRC	DSP	DSP
Internal reconfiguration				
Tufte and Haddow (2000)	FIR filters	Register values	HW	HW
Glette and Torresen (2005)	2-bit multipliers	VRC	PowerPC	HW
Martinek and Sekanina (2005)	Image filters	VRC	HW	HW
Sekanina and Friedl (2004)	Arithmetic circuits	VRC	HW	HW
Salomon et al. (2006)	Hash functions	VRC	HW	HW
Glette et al. (2006)	Face recognition	VRC	MicroBlaze	HW
Upegui and Sanchez (2006)	Cellular automaton	Virtex CLB	MicroBlaze	HW
Sekanina et al. (2006)	Polymorphic circuits	VRC	HW	HW

Figure 5 also shows that every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbours in the processed image.

4.2 Reconfigurable processing array

Similarly to Martinek and Sekanina (2005), the reconfigurable image filter will be implemented as a VRC (see Figure 4). As a new pixel value is calculated using nine pixels, the VRC has got nine 8-bit inputs and a single 8-bit output. The VRC consists of two-input CFBs placed in a grid of 8 columns and 4 rows. Any input of each CFB may be connected either to a primary circuit input or to the output of a CFB, which is placed anywhere in the preceding column. Any CFB can be programed to implement one of the functions given in Table 3; all these functions operate with 8-bit operands and produce 8-bit results. These functions were recognised as useful for this task in Sekanina (2004). The reconfiguration is performed column by column. The computation is pipelined; a column of CFBs represents a stage of the pipeline. Registers are inserted between the columns in order to synchronise the input pixels with CFB outputs. The configuration bitstream of VRC consists of 384 bits. A single CFB is configured by 12 bits, 4 bits are used to select the connection of an input, 4 bits are used to select one of the 16 functions. EA directly operates with configurations of the VRC; simply, a configuration is considered as a chromosome.

4.3 Fitness calculation

The fitness calculation is carried out by the Fitness Unit (FU). The pixels of corrupted image u are loaded from external SRAM1 memory and forwarded to inputs of VRC. Pixels of filtered image v are sent back to the Fitness Unit, where they are compared to the pixels of original image w which is stored in another external memory, SRAM2. Filtered image is simultaneously stored into the third external memory, SRAM3. Note that all image data are stored in external SRAM memories due to the limited capacity of internal RAMs available in the FPGA chip.

Table 3 List of functions implemented in a CFB

Code	*Function*	*Description*
0	255	Constant
1	x	Identity
2	$255 - x$	Inversion
3	$x \vee y$	Bitwise OR
4	$\bar{x} \vee y$	Bitwise $\bar{x}$ OR y
5	$x \wedge y$	Bitwise AND
6	not $(x \wedge y)$	Bitwise NAND
7	$x \oplus y$	Bitwise XOR
8	$x \gg 1$	Right shift by 1
9	$x \gg 2$	Right shift by 2
A	$(x \ll 4) \vee (y \gg 4)$	Swap
B	$x + y$	+ (addition)
C	$x +^S y$	+ with saturation
D	$(x + y) \gg 1$	Average
E	$\max(x, y)$	Maximum
F	$\min(x, y)$	Minimum

The design objective is to minimise the difference between the filtered image and the original image. The image size is $m \times n$ pixels but only the area of $(m - 2) \times (n - 2)$ 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 VRC is configured using a candidate chromosome

2 the circuit created is used to produce pixel values in the image v and

3 the fitness value is calculated as

$$\text{fitness} = \sum_{i=1}^{m-2} \sum_{j=1}^{n-2} |v(i, j) - w(i, j)| \tag{1}$$

In order to feed the VRC with 3×3 pixels in every clock cycle, the hardware implementation of the FU utilises three first-in-first-out raw buffers, special addressing circuits and comparators (to detect the end of row and the end of picture). This slightly complicated approach is implied by the fact that the 3×3 pixels of the kernel are not stored at neighbouring addresses of SRAMs.

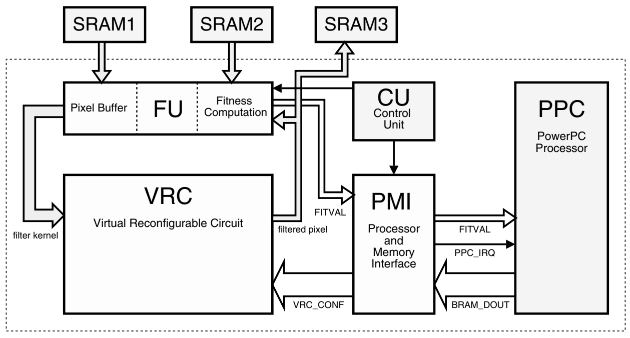
The second part of FU implements the formula given above. As the maximum pixel difference is 255, the biggest number we have to be able to store is 255. For images of 256×256 pixels, the fitness value can be stored using 24 bits.

The FU can be considered as an extension of the VRC pipeline. Hence, in each clock cycle, a temporary fitness value is updated by a new pixel difference.

4.4 Overall architecture

As Figure 6 shows the proposed evolvable system (except the SRAM memories) is completely implemented in a single FPGA. All components (except the VRC) are connected to the LocalBus. Since the previous sections have dealt with the VRC and FU, it remains to describe the Control Unit (CU), Processor and Memory Interface (PMI) and the PowerPC integration into the system.

Figure 6 Overall architecture of the image filter evolution. SRAMs are utilised to store input and output images



The CU is a hardwired controller which plays the role of master and controls the entire system. In particular, it starts/stops the evolution, determines the number of generations and other parameters of EA and generates the control signals for the remaining components. Considering the throughputs of buses and interfaces, this design alternative represents the most flexible and powerful solution.

PowerPC is considered as a component that is able to generate a new candidate individual when it is requested. In other words, it is idle in its main loop. Program memory of the PowerPC is implemented using on-chip BRAM memories and connected to the LocalBus in order to send/read programs to/from an external PC which is connected with FPGA via a PCI bus. Since our program is short, it can completely be stored in an instruction cache.

The population of candidate configurations is stored in on-chip BRAM memories. The population memory is divided into banks; each of them contains a single configuration bitstream of VRC. An additional bit (associated with every bank) determines data validity; only valid configurations can be evaluated. In order to overlap the evaluating of a candidate configuration with generating a new candidate configuration, at least two memory banks have to be utilised. While a circuit is evaluated, a new candidate configuration is generated. A new configuration is utilised immediately after completing the evaluation of the previous circuit. If b banks are utilised, the PowerPC processor has b-times more time to generate a new candidate circuit (i.e. EA can be more complicated). The proposed implementation utilises eight banks.

The PMI component consists of two subcomponents working concurrently. The first subcomponent, controlled by the CU, reconfigures the VRC using configurations stored in the population memory. The second subcomponent is responsible for sending the fitness value to the PowerPC processor. This process is controlled by the FU. The PMI component also provides an interface to the population memory (BRAMs) via LocalBus.

The evaluation of candidate configurations works as follows:

1. When a valid configuration is available, the CU initiates the reconfiguration of VRC. This process is controlled by PMI.
2. As soon as the first column of CFBs has been reconfigured, CU initiates the fitness calculation process performed by the FU.
3. When the last column of CFBs has been reconfigured, a corresponding memory bank is invalidated and the bank counter is incremented.
4. Three clock cycles before the end of evaluation the FU indicates the forthcoming end of evaluation.
5. The CU initiates a new configuration of VRC and repeats the sequence 1–4 again.
6. As soon as the fitness value is valid, it is sent (together with a corresponding bank number) to the PowerPC. An interrupt (IRQ) is generated to activate a service routine of the PowerPC. In this routine, a new candidate configuration is generated for the given bank. The PowerPC processor acknowledges the interrupt (IRQACK) and sets up the validity bit.

These steps are pipelined in such manner as there are no idle clock cycles. Therefore, the time of a candidate circuit evaluation can be expressed as

$$t_{\mathrm{eval}} = (m-2)(n-2)\frac{1}{f} = (m-2)(n-2)\frac{1}{50}\mu\mathrm{s}$$

where $n \times m$ is the number of pixels and f is the operation frequency. Table 4 gives the evaluation time for different sizes of images and $f = 50$ MHz.

Table 4 The evaluation time and the number of evaluations that can be performed within 1 sec ($f = 50$ MHz)

Image size	*Evaluation time*	*Evaluations per second*
32×32	18 μs	55,555
64×64	77 μs	13,007
128×128	318 μs	3149
256×256	1291 μs	775

4.5 Results of synthesis

In order to implement the proposed system, we used a COMBO6X card developed in Liberouter project (2005). The evolvable system was described in VHDL, simulated using ModelSim and synthesised using Mentor Graphics Precision RTL and Xilinx ISE tools to Virtex II Pro 2VP50ff1517 FPGA. Results of synthesis are summarised in Table 5. The whole design occupies approximately 20% of the FPGA. The VRC represents approximately 75% of the design. While the PowerPC works at 300 MHz, the logic supporting the PowerPC works at 150 MHz. The remaining FPGA logic (including VRC and FU) works at 50 MHz.

Table 5 Results of synthesis for the Virtex II Pro 2VP50ff1517 FPGA

VRC	*IO blocks*	*BRAM*	*CLB*	*DFF*
Available	852	232	23,616	49,788
4 × 8 CFBs	602	12	4591	3638
used	70%	5%	20%	7%
No VRC	602	12	1240	2479
used	70%	5%	5%	5%

5 Experimental results

5.1 Parameters of experiments

This section reports results of experiments arranged with the aim of comparing the proposed implementation with previous implementations of Martinek and Sekanina (2005). In all experiments, we utilised training images of 128 × 128 pixels and allowed to perform up to 49,512 evaluations (which corresponds with approximately 15 sec of evolution). The population contains eight individuals. In all experiments, the mutation operator inverts seven randomly selected bits. These parameter values were experimentally found to be suitable for the proposed comparisons.

5.2 Test problems

The platform is tested on two problems:

- The design of a shot noise removal filter. Here, pixels are corrupted with the shot noise (a pixel value set at maximum) with the probability of 5%. In conventional implementations, this noise is suppressed by the median filter.
- The design of an edge detection operator. Various edge detectors exist, see Sonka et al. (1999). Our results will be compared with the Sobel operator.

5.3 Search algorithms

As the search algorithm is stored in the program memory of the PowerPC processor, the proposed platform allows the designer to easily modify the search algorithm. Three search algorithms are evaluated: a Random Search (RS), a Hill Climbing (HC) algorithm and a Genetic Algorithm (GA).

RS: by analysing the search algorithm implemented as a special digital circuit in Martinek and Sekanina (2005), we recognised that a parallel RS was actually implemented. This algorithm operates with eight individuals that are generated randomly at the beginning of the evolution. Then an offspring is created using a bit-mutation operator from each parent and evaluated. If the offspring is equal or better than its parent then the offspring replaces the parent in the new population.

HC algorithm: this algorithm operates with eight individuals that are generated randomly at the beginning of the evolution. After their evaluation, eight offspring configurations are generated for each parent using a bit-mutation operator. The best offspring of the eight offspring configurations replaces the corresponding parent; however, only in case that its fitness value is equal or better than the parent's fitness value.

GA: the initial population of p individuals is generated randomly. Then, k offspring are generated from each parent using a bit-mutation operator. A new population consisting of p individuals is formed from p parents and their pk offspring. We utilised a deterministic selection in which p-best scored individuals are selected as new parents. In our experiments, $p = 8$ and $k = 8$. No crossover operator is utilised because it is currently unknown how to design it to be more efficient than the mutation operator.

5.4 Comparison of results

Table 6 summarises results obtained for the three search algorithms and the two test problems. 100 independent runs were performed for each problem and 49,512 fitness evaluations were allowed in a run.

The results are given for a training image. While shot noise filters were evolved using an alloy image (taken from a microscope), a Lena image was utilised for edge detectors. As the fitness value expresses the difference between filtered and reference images, the lower value the better result. The GA performs significantly better on the both problems. The average fitness values suggest that the shot noise filter design task is easier than the edge detector design task.

Figure 7 shows a three-dimensional normalised histogram of resulting fitness values calculated from 100 independent runs. For GA, it can be seen that the resulting fitness values are concentrated close to lower values.

In order to illustrate differences between the three algorithms and between the average filters and the best-evolved filters, Figure 8 shows a corrupted image and images filtered by some evolved filters. The image filtered by a filter with the average fitness value gives an example of the result which we can obtain with the highest probability at the end of a 15-sec run.

Figure 9 shows results for edge detection. The image obtained by applying the average evolved operator is not sufficient. In order to get a sufficient operator, we had to increase the number of generations to 50,000.

The best-evolved filters were applied to remove the shot noise from images never seen during the learning process. Figure 10 demonstrates that the evolved filters are operating correctly for a certain class of images. Note that we also tested those filters on 256 × 256-pixel images although they were trained on 128 × 128-pixel images. Similarly to observations

Table 6 Fitness values for three search algorithms. Averages were calculated from 100 runs

Algorithm/fitness Value	*Minimum*	*Maximum*	*Average*	*SD*
Genetic Algorithm (GA)	8312	150,415	18,625	16,196
Hill Climbing (HC)	10,138	202,176	50,357	42,691
Random Search (RS)	10,683	152,775	33,226	24,776
Genetic Algorithm (GA)	118,001	453,609	298,995	70,494
Hill Climbing (HC)	136,780	544,112	346,682	82,137
Random Search (RS)	138,952	632,796	331,599	81,793

Figure 7 Normalised histograms of resulting fitness values for 100 runs (a) shot noise and (b) Sobel operator

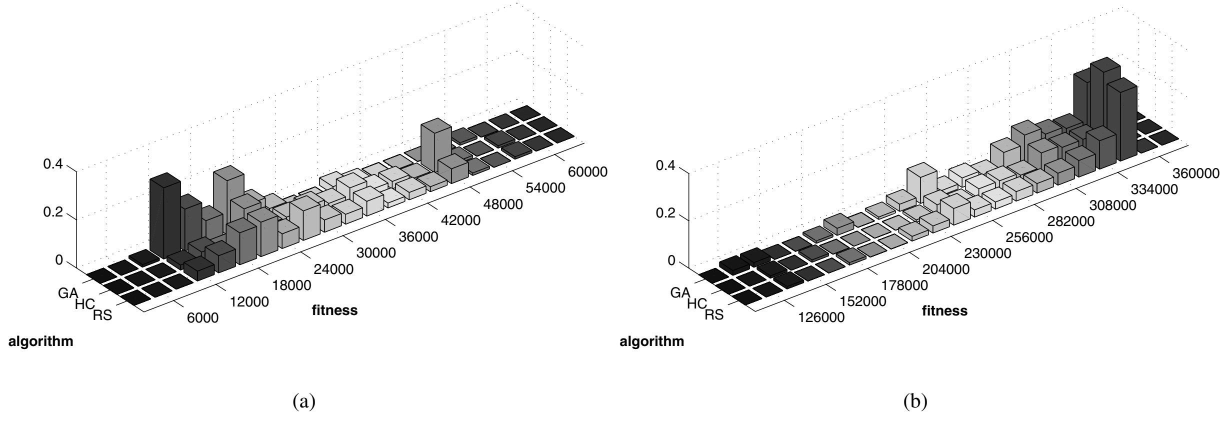


Figure 8 A corrupted image containing the shot noise (a), a reference image (e) and images filtered using some of evolved filters: (a) input; (b) average RS; (c) average HC; (d) average GA; (e) required output; (f) the best RS; (g) the best HC and (h) the best GA

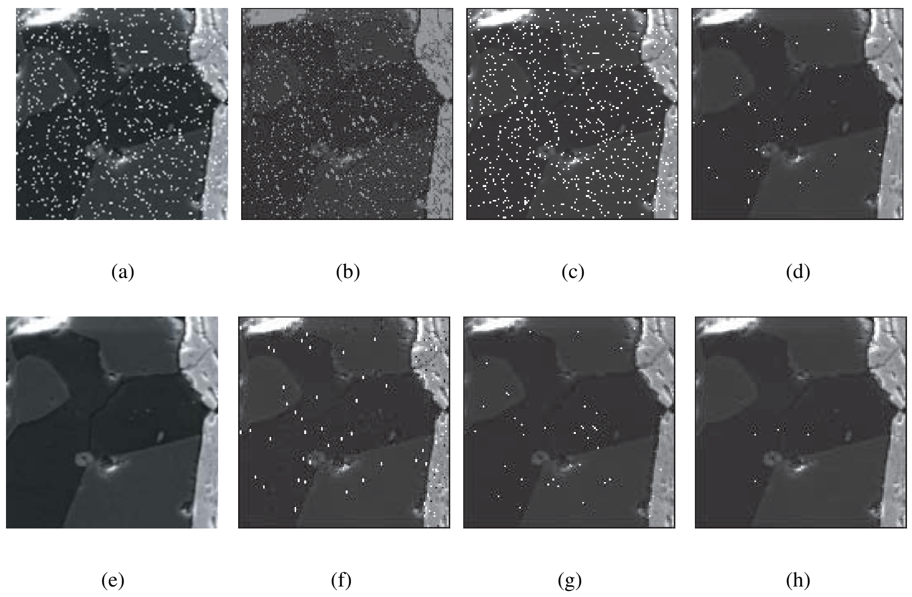


Figure 9 Examples of images filtered using edge detectors: (a) input; (b) required output; (c) average GA and (d) the best GA

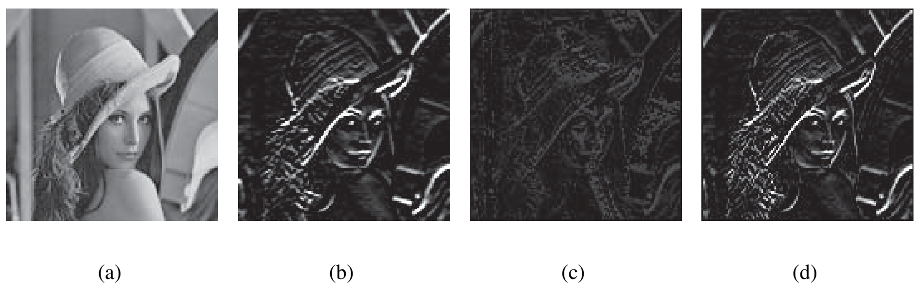
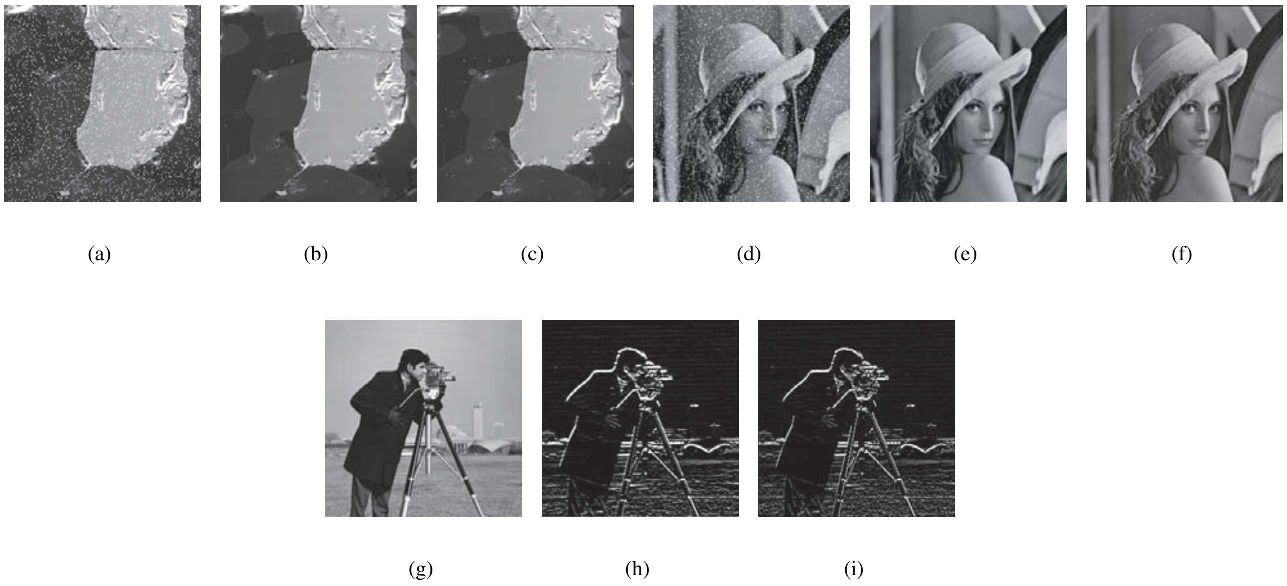


Figure 10 Responses of some of evolved filters on the images from a test set: (a) input; (b) required output; (c) filtered image; (d) input; (e) required output; (f) filtered image; (g) input; (h) required output and (i) filtered image



reported in Martinek and Sekanina (2005), we can confirm that images filtered by evolved filters exhibit more details (and thus a higher visual quality) than images filtered by conventional filters (e.g. median filters).

The following C program represents an implementation of the best-evolved edge detector created according to Figure 11. Note that *kernel* represents the nine inputs of the image filter.

```
uint8 filter(uint8 kernel[9]) {
  uint i14,i17,i19,i22,i27,i29;

  i14 = min((kernel[1] + kernel[7])
  >> 1, kernel[7]);
  i17 = i14 ^ kernel[7];
  i19 = min(i14+(255 - kernel[1]), 255);
  i22 = 255 - i19;
  i27 = min(i22, (i17 + i19) >> 1);
  i29 = min(i22 + i27, 255);
  return (i27 + i29) & 0xff;
}
```

In contrast, the following C code represents a conventional implementation of the Sobel edge detector which was used to construct a target image for the fitness function.

```
uint8 filter(uint8 kernel[9]) {
  int i;

  i = kernel[0]+ 2*kernel[1]+ kernel[2];
  i = i - (kernel[6] + 2*kernel[7]
  + kernel[8]);
  i = max(i, 0);
  i = min(i, 255);
  return i;
}
```

The goal of the final set of experiments is to search for as good filter as possible in case that many populations can be generated. Filters are evolved for a 128 $\times$ 128 Lena image corrupted by shot noise. One hundred independent runs are performed with the maximum number of 50,000 generations. Other parameters remain unchanged. Table 7 shows that average fitness values are considerably lower than those reported in Table 6. The best evolved filter exhibits the mean difference per pixel mdpp $= 0.26$ (between filtered and reference images). The best mdpp known for this type of noise and Lena image is 0.38 (see Sekanina and Ruzicka, 2003).

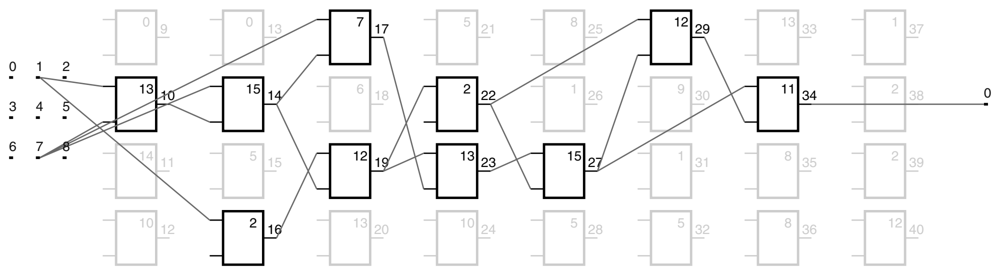
Table 7 Fitness values for RS and GA obtained after 50,000 generations

Fitness Value	*Minimum*	*Maximum*	*Average*	*SD*
GA	4153	50,332	9937.9	10640.5
RS	5325	24,548	10233.3	3851.5

6 Discussion

The solution based on the PowerPC processor benefits from a software implementation of the search algorithm allowing the user to easily modify and tune the search process for a particular problem. Firstly, we evaluated the RS algorithm which is a software version of the complete circuit implementation reported in Martinek and Sekanina (2005). For image filter design, this complete circuit implementation exhibits the speedup of 50 in comparison with a software simulator running at common PC. As the proposed implementation as well as the complete circuit implementation utilises the identical VRC, operates at the same frequency (50 MHz) and there are no idle clock cycles, we can consider them as equivalent. We verified that the both implementations perform the same number of evaluations for a given time limit and that the visual quality of filtered images is indifferentiable. While the RS algorithm requires 117,552 evaluations in average, GA requires only 49,512 evaluations

Figure 11 The best edge detector evolved in reported experimental results. Note that functions in CFBs are numbered according to Table 3



in average to obtain filters generating output images of the same visual quality (see e.g. in Figure 10(f)). Corresponding mean differences per pixel are 1.2 for GA and 1.1 for RS (for Lena image and shot noise). This means that the design process is 2.37 times faster in average in case of GA. Therefore, by using a more sophisticated search algorithm, we are able to improve the performance of the system.

Note that HC is the worst one of the compared algorithms. That is probably because the corresponding search spaces are very rugged and HC traps in a local optimum quickly.

In order to compare the implementation cost, we have to look at results of synthesis. Table 8 compares resources utilised for the implementation with the PowerPC processor and the complete circuit implementation without the PowerPC processor. Although different FPGAs are utilised, the comparison is quite fair because the building blocks are identical. We can observe that both implementations require similar resources (if the PowerPC core is not taken into account). Roughly speaking, the resources utilised for the EA in the original implementation were used to build supporting logic (PMI controller, caches, etc.) for the PowerPC.

Table 8 Comparison of resources utilised for the implementations with and without the PowerPC processor (for 4 × 8 CFBs in the VRC and an eight-member population)

Solution	*Virtex II FPGA*	*BRAMs*	*Func. geners.*	*DFF*
With PowerPC	Pro 2VP50ff1517	12	9181	3638
No PowerPC	XC2V3000bf957	2	10,331	3104

Providing a corrupted image and a corresponding original image, the proposed system is able to generate an image filter which is highly competitive with filters designed conventionally and utilised in practice. The resulting filter can be generated in 10 sec. In this paper, the approach was verified for two types of noise; however, Sekanina (2004), Sekanina and Ruzicka (2003) and Martinek and Sekanina (2005) have shown its effectiveness on a relatively broad class of noise types.

The design time is very reasonable if the proposed system should operate 'instead' of a human designer and the resulting filter should be offered as a product on a market independent of the fact that the filter was mechanically created. Further speedup is possible by introducing a higher degree of parallel processing, for example, if more than one VRC were implemented. For some applications, our solution could also operate as a real-time evolving and adaptive filter. As the proposed solution is designed as a system on a chip, it is also suitable for various (small) embedded systems.

7 Conclusion

In this paper, a new architecture for image filter evolution was proposed and evaluated. The evolvable system is based on a software implementation of the search algorithm in the PowerPC processor which is available in Xilinx Virtex II and 4 FPGAs. Candidate filters are evaluated in a domain-specific virtual reconfigurable circuit implemented using a reconfigurable logic of the same FPGA. As the PowerPC processor enables implementing more sophisticated search algorithms than an original circuit solution, a higher performance can be obtained. The proposed evolvable image filter is one of many applications that can be developed using a modular architecture that we created in a commercial off-the-shelf FPGA. By modifying the VRC and the search algorithm running in the PowerPC processor, the FPGA can be utilised to effectively evolve other digital circuits, such as predictors, controllers, classifiers, hash functions and unconventional operators, whose design is difficult for a human designer.

Acknowledgements

This work has been financially supported by the Grant Agency of the Czech Republic under No. 102/07/0850 'Design and hardware implementation of a patent-invention machine' and the Research Plan No. MSM 0021630528 'Security-Oriented Research in Information Technology'.

References

Atmel Corp. (2002) 'Atmel AT94K Series FPSLIC', Available at: http://www.atmel.com/dyn/resources/prod_documents/1138S. pdf.

Blodget, B., James-Roxby, P., Keller, E., McMillan, S. and Sundararajan, P. (2003) 'A self-reconfiguring platform', *Proceedings of the 13th Conference on Field Programmable Logic and Applications FPL'03*, Volume 2778 of *LNCS*, Lisbon, Portugal: Springer Verlag, pp.565–574.

Burian, A. and Takala, J. (2004) 'Evolved gate arrays for image restoration', *Proceedings of 2004 Congress on Evolutionary Computing CEC'04*, IEEE Publication Press, pp.1185–1192.

de Garis, H. (1993) 'Evolvable hardware – genetic programming of a darwin', *International Conference on Artificial Neural Networks and Genetic Algorithms*, Innsbruck, Austria: Springer Verlag.

Dumoulin, J., Foster, J., Frenzel, J. and McGrew, S. (2000) 'Special purpose image convolution with evolvable hardware. *Real-World Applications of Evolutionary Computing – Proceedings of the second Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'00*, volume 1803 of *LNCS*, Springer Verlag, pp.1–11.

Erba, M., Rossi, R., Liberali, V. and Tettamanzi, A. (2001) 'An evolutionary approach to automatic generation of VHDL code for low-power digital filters', *Proceedings of the Fourth European Conference on Genetic Programming EuroGP2001*, volume 2038 of *LNCS*, Springer Verlag, pp.36–50.

Glette, K. and Torresen, J. (2005) 'A flexible on-chip evolution system implemented on a xilinx virtex-ii pro device', *Evolvable Systems: From Biology to Hardware*, volume 3637 of *LNCS*, Springer Verlag, pp.66–75.

Glette, K., Torresen, J., Yasunaga, M. and Yamaguchi, Y. (2006) 'On-chip evolution using a soft processor core applied to image recognition. *The First NASA/ESA Conference on Adaptive Hardware and Systems*, Los Alamitos, CA: IEEE Computer Society, pp.373–380.

Gordon, T. (2005) 'Exploiting development to enhance the scalability of hardware evolution', PhD Thesis, Department of Computer Science, University College London.

Gwaltney, D. and Dutton, K. (2005) 'A VHDL core for intrinsic evolution of discrete time filters with signal feedback', *Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*, Washington D.C: IEEE Computer Society, pp.43–50.

Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H. and Furuya, T. (1993) 'Evolving hardware with genetic learning: a first step towards building a darwin machine', *Proceedings of the Second International Conference on Simulated Adaptive Behaviour*, MIT Press, pp.417–424.

Hollingworth, G., Tyrrell, A. and Smith, S. (1999) 'Simulation of evolvable hardware to solve low level image processing tasks', *Proceedings of the Evolutionary Image Analysis, Signal Processing and Telecommunications Workshop*, volume 1596 of *LNCS*, Springer Verlag, pp.46–58.

Huelsbergen, L., Rietman, E. and Slous, R. (1999) 'Evolving oscillators in silico', *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 3, pp.197–204.

Liberouter project (2005) 'Liberouter project', Available at: http://www.liberouter.org.

Martinek, T. and Sekanina, L. (2005) 'An evolvable image filter: experimental evaluation of a complete hardware implementation in fpga. *Evolvable Systems: From Biology to Hardware*, volume 3637 of *LNCS*, Springer Verlag, pp.76–85.

Porter, P. (2001) 'Evolution on FPGAs for feature extraction', PhD Thesis, Queensland University of Technology, Brisbane, Australia.

Salomon, R., Widiger, H. and Tockhorn, A. (2006) 'Rapid evolution of time-efficient packet classifiers. *IEEE Congress on Evolutionary Computation*, Vancouver, Canada: IEEE CIS, pp.2793–2799.

Sekanina, L. (2002) 'Image filter design with evolvable hardware', *Applications of Evolutionary Computing – Proceedings of the Fourth Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02*, volume 2279 of *LNCS*, Kinsale, Ireland: Springer Verlag, pp.255–266.

Sekanina, L. (2004) *Evolvable Components: From Theory to Hardware Implementations Natural Computing*, Berlin: Springer Verlag.

Sekanina, L. and Friedl, S. (2004) 'An evolvable combinational unit for fpgas', *Computing and Informatics*, Vol. 23, No. 5, pp.461–486.

Sekanina, L. and Ruzicka, R. (2003) 'Easily testable image operators: the class of circuits where evolution beats engineers', *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, USA: IEEE Computer Society, pp.135–144.

Sekanina, L., Martinek, T. and Gajda, Z. (2006) 'Extrinsic and intrinsic evolution of multifunctional combinational modules', *IEEE Congress on Evolutionary Computation*, Vancouver, Canada: IEEE CIS, pp.9676–9683.

Smith, S., Leggett, S. and Tyrrell, A. (2005) 'An implicit context representation for evolving image processing filters', *Applications of Evolutionary Computing*, volume 3449 of *LNCS*, Lausanne: Springer Verlag, pp.407–416.

Sonka, M., Hlaváč, V. and Boyle, R. (1999) *Image Processing: Analysis and Machine Vision*, Thomson-Engineering.

Stoica, A. (2004) 'Evolvable hardware for autonomous systems', *CEC Tutorial*.

Thompson, A., Layzell, P. and Zebulum, S. (1999) 'Explorations in design space: unconventional electronics design through artificial evolution', *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 3, pp.167–196.

Tufte, G. and Haddow, P. (2000) 'Evolving an adaptive digital filter', *The Second NASA/DoD workshop on Evolvable Hardware*, Palo Alto, CA: IEEE Computer Society, pp.143–150.

Upegui, A. and Sanchez, E. (2006) 'Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs', *The First NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2006)*, Los Alamitos, CA: IEEE Computer Society, pp.153–160.

Xilinx Inc. (2005) 'Xilinx Virtex-II Pro Platform FPGAs: complete data sheet', Available at: http://www.xilinx.com/partinfo/ds031.pdf.

Xilinx Inc. (2006) 'Xilinx MicroBlaze Processor Reference Guide', Available at: http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf.

Zhang, Y., Smith, S. and Tyrrell, A. (2004a) 'Digital circuit design using intrinsic evolvable hardware', *Proceeding of the 2004 NASA/DoD Conference on Evolvable Hardware*, Seattle, USA: IEEE Computer Society, pp.55–62.

Zhang, Y., Smith, S. and Tyrrell, A. (2004b) 'Intrinsic evolvable hardware in digital filter design', *Applications of Evolutionary Computing*, volume 3005 of *LNCS*, Quimbra, Portugal: Springer Verlag, pp.389–398.