Published in IET Computers & Digital Techniques
Received on 30th August 2007
Revised on 23rd December 2007
doi: 10.1049/iet-cdt:20070124

ISSN 1751-8601

Design and implementation of a virtual reconfigurable architecture for different applications of intrinsic evolvable hardware

J. Wang   Q.S. Chen   C.H. Lee

Department of Information and Communication Engineering, Inha University, Incheon, Republic of Korea
E-mail: wangjin_liips@yahoo.com.cn

Abstract: The authors present a novel virtual reconfigurable architecture (VRA) for realising real-world applications of intrinsic evolvable hardware (EHW) on field programmable gate arrays (FPGAs). The phenotype representation of the proposed evolvable system is based on a two-dimensional function element (FE) network. Compared with the traditional Cartesian genetic programming, the proposed approach includes more connection restrictions in the FE network to reduce genotype length. Another innovative feature of the VRA is that the whole evolvable system, which consists of an evolutionary algorithm unit, a fitness value calculation unit and an FE array unit, can be realised on a single FPGA. On this work, a custom Xilinx Virtex xcv2000E FPGA, which is fitted in the Celoxica RC1000 Peripheral Component Interconnect (PCI) board is utilised as the hardware platform. The main motive of the research is to design a general, flexible evolvable system with powerful computation ability to achieve intrinsic evolution. As examples, the proposed evolvable system is devoted to evolve two real-world applications: a character recogniser and an image operator by using gate level evolution and function level evolution, respectively. The experimental results show that the VRA can bring higher computational ability and more flexibility than traditional approach to intrinsic EHW.

1 Introduction

Inspired by natural evolution, evolvable hardware (EHW) was developed in the early 1990s as a new concept in the development of adaptive machines. In contrast to the traditional hardware where structure and functions are irreversibly fixed once implemented, EHW refers to a type of hardware whose architecture and functions can change dynamically and autonomously by interacting with its environment. This adaptation ability of EHW, achieved by employing evolutionary learning and reconfigurable device, has great potential for the development of innovative and powerful real-world applications.

Early EHW experiments simulated evolution by software and downloaded only an elite chromosome to hardware (i.e. the hardware was reconfigured only once), which was called extrinsic EHW. In recent years, with the advent and development of software-reconfigurable logic device [e.g. programmable logic devices and field programmable gate arrays (FPGAs)], the idea of intrinsic EHW that is directly evaluating each candidate circuit in every population in a physical reconfigurable device has been successfully realised [1]. In the intrinsic EHW approach, one of the most interesting features is that the calculation of fitness value of each candidate circuit, which is the most time-consuming part in the whole evolutionary process, can be evaluated more quickly than in software-simulation-based extrinsic EHW. Another visible advantage of intrinsic EHW is that it can exploit physical properties of the electronic platform or environment (e.g. temperature [2], power supply voltage [3] etc.) and thus to provide some innovative features.

In this paper, we propose a flexible intrinsic EHW platform named virtual reconfigurable architecture (VRA), which explores the idea of virtual reconfiguration allowing the designers to perform different high-performance evolvable applications on commercial FPGAs. The novelty of our proposed VRA lies in its complete FPGA implementation of function element (FE) array, fitness calculation and evolutionary algorithm (EA) with the features of pipelining and inner parallelisation, which is presented as a strategy for intrinsic EHW. The evolvable system is programmed with hardware description language (HDL) source code, which means the VRA can routinely be realised in various FPGA platforms. A new genotype–phenotype representation scheme is employed to release the issue of high hardware cost, which has appeared in some reports of Cartesian genetic programming (CGP) [4] inspired virtual reconfigurable platforms [5–7].

The objective of this paper is to design, implement and evaluate this VRA for different real-world applications. As a case study to illustrate the power and flexibility of our proposed method, both of gate level evolution of a character recognition system and function level evolution of a spatial image operator are implemented in our experiments. A lot of related work have been done in the area of evolving character recognition systems [8–10] and spatial image filter [6, 11–13] because of their potential merit in research and industry. Experimental results show that the proposed scheme can evolve target circuits more efficiently and flexibly than other reported extrinsic and intrinsic EHW approaches.

The rest of this paper is organised as follows: Section 2 discusses the contemporary intrinsic EHW techniques. The idea of VRA is detailed in Section 3. In Section 4 we present the FPGA implementation of the VRA. The experimental results of evolving character recognition system and spatial image filter are summarised in Section 5. Section 6 contains a discussion about the advantages and the limitations of the proposed approach. Finally, Section 7 concludes this paper.

2 Intrinsic evolution on FPGAs

To speed up fitness evaluation, with the rapid development of reconfigurable devices, different intrinsic EHW approaches have been proposed in recent years. FPGAs which enjoy both the high performance of a dedicated hardware solution and the flexibility of software offered by its inherent reprogram ability feature are the most popularly utilised commercial reconfigurable logic device for digital intrinsic EHW. A number of works have been done in the area of FPGA-based intrinsic EHW. Generally, these approaches can be divided into two groups:

2.1 FPGA configuration bitstream-based intrinsic EHW

The first kind of approach to intrinsic EHW is to regard the configuration bitstream of FPGA as EA chromosome. Every chromosome generated by EA can be downloaded to FPGA via an external configuration port (e.g. via JTAG connector, SelectMAP interface etc.) for fitness evaluation. Xilinx XC6200 series FPGAs [14] were the first widely used commercial reconfigurable device in EHW research for their evolution-friendly features such as partial reconfiguration, open bitstream format, protection against illegal configurations etc. A well known example using Xilinx XC6200 series is that of Thompson who has successfully designed an EHW platform to evolve tone discriminator circuits [15]. With the demise of Xilinx XC6200 series chips, more recent works have focused on Xilinx Virtex family [16]. In Tyrrell's group, they have successfully employed Jbits [17], a set of Java classes, which provide an application program interface for describing circuits and manipulating the bitstream of Virtex FPGA, to evolve some simple circuits [18, 19]. Dynamic partial reconfiguration (DPR) technique [20] is another potential approach to intrinsic EHW on Virtex FPGAs. In [21], Upegu and Sanchez have implemented a DPR-based intrinsic EHW to evolve a random Boolean network.

Even though the idea of direct evolution of configuration bitstreams of FPGAs provides a promise for realising intrinsic EHW, it has several limitations: (1) Because the FPGA configuration bitstream is directly encoded as EA chromosome, its granularity is too fine to evolve complex circuits (i.e. it generally makes the chromosome length to be on the order of tens of thousands of bits, rendering evolution practically impossible using current EA technology). (2) In today's digital EHW, under the demand for 10 000 or even more than 100 000 of FPGA reconfigurations to evolve a small result circuit (e.g. a 3 bit multiplier, a 3 bit adder etc.), the reconfiguration time of FPGA becomes an obvious bottleneck for intrinsic EHW. Table 1 [22] shows some typical reconfiguration times of Virtex FPGAs when the whole chip is reconfigured. Partial reconfiguration technology seems to be a potential solution to this problem, because a shorter timing requirement can be achieved by making small modification without having to reconfigure the entire FPGA device. Even if partial reconfiguration techniques provide some interesting features, they have several limitations, such as the perfect platform Xilinx XC6200 has become obsolete for commercial reasons, Jbits and DPR techniques are depended on supported FPGA families and generally too complicated for most of users. On the other hand, in most of partial reconfiguration-based applications, the reconfiguration timings around 1–100 ms are common (see Table 1; even though we modify only the configuration of one CLB column, all the required configuration timings in

Table 1 Configuration time of a CLB column and the whole device

FPGA family	One CLB column, ms	Whole device, ms
XCV50	1.228	37.282
XCV400	2.560	169.738
XCV2000E	4.915	677.309

different FPGAs are more than 1 ms). It is still too slow to achieving more than 100 000 of reconfigurations in a reasonable time.

2.2 Virtual reconfiguration-based intrinsic EHW

Instead of directly manipulating FPGA configuration bitstream as an EA chromosome, an alternative approach to intrinsic EHW is virtual reconfiguration technique, which employs a higher-level representation to specify the target circuit architecture and function. With this scheme, FPGA cells are grouped together into some subsets of cells as an FE network and newly defined configuration bits are used to determine the functions and interconnections of these FEs. In this approach, the normal FPGA configuration bitstream should not be changed during evolution. The evolution is instead applied on the newly defined configuration bits of the FE network. The original idea of virtual reconfiguration has been proposed by different researchers, such as two stage approach to reconfiguration developed by Porter *et al.* [23], Slorach and Sharman's customer-built programmable devices [24], virtual EHW FPGA suggested by Haddow and Tufte [25] and the virtual reconfigurable circuit introduced by Sekanina [7]. A broad range of real-world applications of virtual reconfiguration-technique-based intrinsic EHW may be found in the recent published literatures. Zhang *et al.* introduced the evolution of spatial image filters on Xilinx Virtex FPGA xcv1000 [11]. For the evolutions of 2 bit multiplier [26] and image recognition system [27], Glette and Torresen built their on-chip intrinsic evolution platforms on the top of Xilinx Virtex-II Pro series FPGAs, which include PowerPC 405 embedded processors to implement EA. It is also worth mentioning Sekanina's implementation. On a specialised FPGA platform-COMBO6 card, he has achieved complete hardware evolution of digital image operators [6] with virtual reconfigurable circuit.

3 Virtual reconfigurable architecture

In this paper, we develop and design a virtual reconfiguration-technique-based EHW platform, named VRA.

3.1 Basic concept

The VRA, which is described in an HDL, is a second reconfiguration layer developed on the top of an FPGA. The key goals of our proposed VRA are to provide a much simpler intrinsic EHW platform – thus reducing the length of the genotype description and providing the feature of fast internal reconfiguration. Fig. 1 shows that our VRA consists of an FE array and an EA. The top function of the FE array is configured using the chromosomes generated by EA, which is implemented on the same FPGA. The chromosome encodes the functions performed by each FE and the interconnection of the FE array. As the fitness calculation is also carried out in the same FPGA, we can benefit from pipeline processing allowing reasonable time of a candidate circuit evaluation. From this perspective, the approach utilising a VRA offers many benefits, including: (1) In the VRA, the FE array is directly connected to the hardware implementation of the EA placed on the same FPGA allowing the bottleneck introduced by slow communication between FPGA and

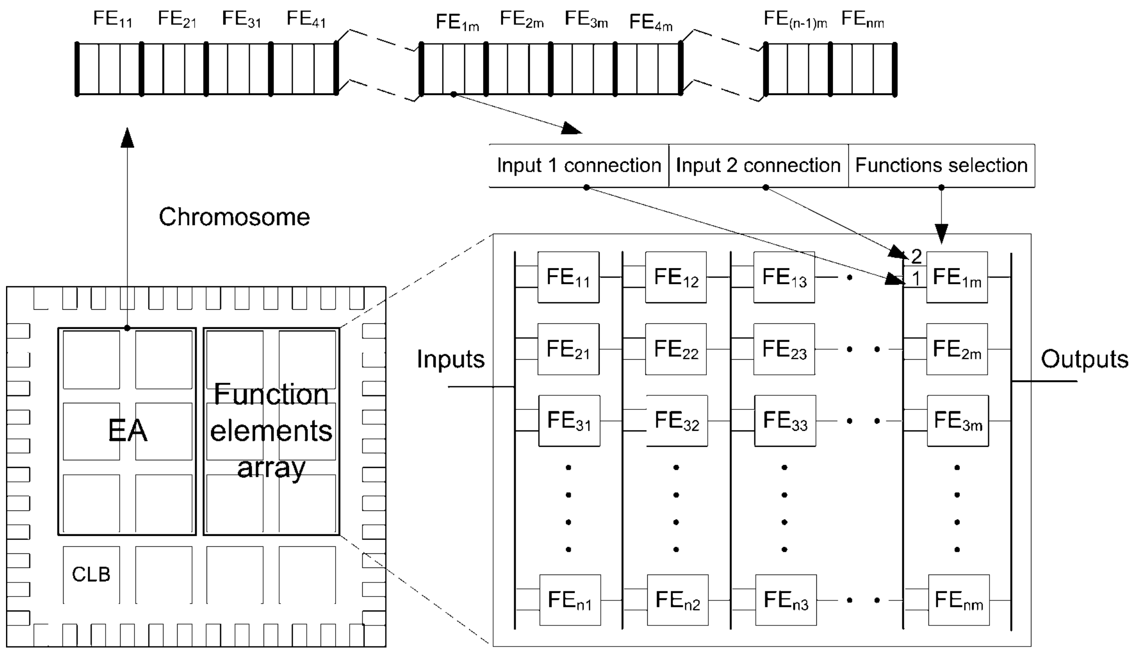

Figure 1 *VRA and its genotype–phenotype mapping*

PC can be overcome. (2) Because the VRA is available at the level of HDL source code, it can easily be modified and synthesised for various target platforms. (3) The VRA can be designed exactly according to the requirements of a given problem. This feature means that the granularity of the FE array can exactly fit the needs of a given application.

3.2 Genotype–phenotype mapping

In this paper, the encoding of a digital circuit into a genotype is a development of earlier models of Wang *et al.* [10, 12], Torresen [9, 26] and Coello *et al.* [28]. As shown in Fig. 1, the phenotype is a connected two-input FE network in the two-dimensional FE array. This FE array consists of m columns from system input to output in which any FE may be either connected or disconnected. Each FE in column l can receive its two inputs from any FE in column $l-1$. Each FE can have one of the functions predefined in a function set (such as AND, OR, XOR, addition, subtraction etc.).

The genotype is a fixed-length linear binary bit string, which defines the interconnections of FEs and functions performed by FEs. The mapping process of the genotype to the phenotype is also illustrated in Fig. 1. The chromosome string encodes the FE array by using triplets in which the first two blanks refer to each of the FE's inputs employed, and the third is the achieved function from the available functions set. Compared with CGP employed by Sekanina [5, 6, 29] and Zhang *et al.* [11] in their virtual-reconfiguration-based intrinsic EHW in which an FE's input can be connected to the output of some FEs in its preceding columns or to some of the system inputs, our proposed two-dimensional geometric structure restricts the inputs of each FE to come only from the FEs in its previous one column to achieve the purpose of reducing genotype length.

Similar to CGP, an obvious advantage of the proposed two-dimensional geometric-structure-based representation is that the genotype representation used is independent of the data type used in the phenotype. This feature brings us a generic and flexible platform. As for different applications, one would just need to change the data type, leaving the genotype unchanged. For instance, by changing the function set and the size of datapaths in the FE array, our proposed model may be suitable for both gate level and function level evolutions with a comparable chromosome size.

3.3 Evolutionary algorithm

In this paper, the EA used to produce all of the applications is a simple form based on the $1+\lambda$ evolutionary strategy [30], where $\lambda = 4$. The employed EA is only based on selection and mutation operators (MOs). No crossover is applied because several previous experiments [4, 11, 29] have indicated that the crossover operators do not significantly improve the quality of the search. The flow diagram of the employed EA is as follows:

1. Randomly initialises a population of λ individuals.

2. Evaluates all individuals.

3. Find the best individual.

4. Create λ mutants of the best individual.

5. Create a new $1+\lambda$ population using λ mutants and the best individual.

The algorithm repeats Steps 2–5 until a termination condition is achieved. The termination condition of EA in this paper is defined as: (1) the predefined EA generation number is exhausted; or (2) EA finds the expected solution.

4 Implementation of VRA

A Celoxica RC1000 PCI board, which has been successfully applied as a high performance, flexible and low-cost FPGA platform for different computational-intensive applications [31, 32], is employed as our target hardware for the implementation and verification of the proposed VRA. The organisation of the VRA on Celoxica RC1000 PCI board is given in Fig. 2. Celoxica RC1000 PCI board includes a Xilinx Virtex xcv2000E FPGA and has 8 Mb of static random access memory (SRAM), which is directly connected to the FPGA in four 32 bit wide memory banks. All memory banks are accessible by the FPGA and host PC. There are two methods of data transfer from the host PC to the FPGA existed: (1) two unidirectional 8 bit paths named control and status ports; (2) four 32 bit wide memory banks 0–3.

The front end of the proposed evolvable system consists of a host PC program running on the host PC, which is described with C code and compiled by using MS Visual C++ 6.0 (see system design flow described in Fig. 3).

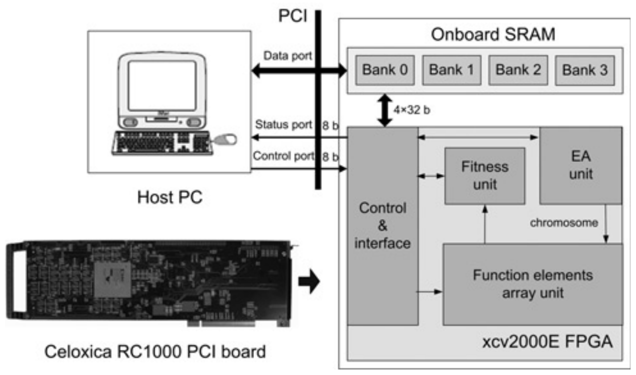

Figure 2 *Hardware organisation of VRA-based evolvable system on Celoxica RC1000 PCI board*

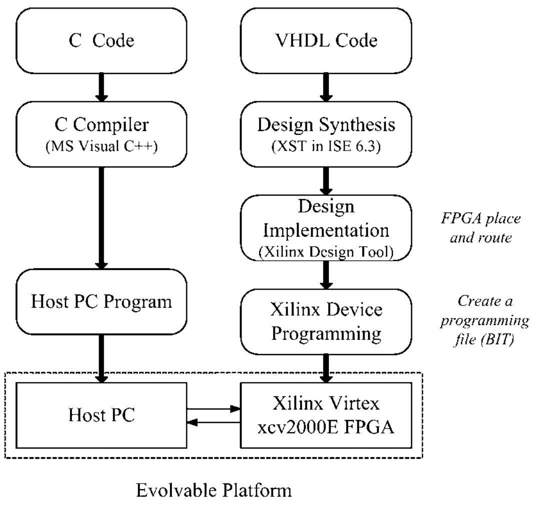
C Code
VHDL Code
C Compiler (MS Visual C++)
Design Synthesis (XST in ISE 6.3)
Design Implementation (Xilinx Design Tool)
FPGA place and route
Host PC Program
Xilinx Device Programming
Create a programming file (BIT)
Host PC
Xilinx Virtex xcv2000E FPGA
Evolvable Platform

Figure 3 *System design flow*

In this work, the host PC program performs the following operations:

1. initialise the RC1000 PCI board;

2. download an FPGA programming file to configure the FPGA;

3. transfer the system inputs set, the expected outputs set and the initial seed (for generating pseudo-random numbers in the evolutionary process) from PC to their corresponding onboard SRAM banks;

4. start the evolution process on FPGA;

5. wait for the FPGA to signal that the evolution has been finished;

6. read and save the results back from the FPGA board.

As seen from Fig. 2, the proposed VRA-based evolvable system on an xcv2000E FPGA is composed of four components: FE array unit, EA unit, fitness unit and control and interface. The VRA is an application-specific system for the evolution of digital combinational circuits. Celoxica RC1000 PCI board supports traditional HDL: for example, VHDL and Verilog. On the other hand, a new C-like system description language called Handel-C introduced by Celoxica [33] is also supported by this board as an alternative. In this paper, VHDL is employed as our design language for its widespread use and standardisation. Xilinx Integrated Software Environment (ISE) 6.3 is used as a design tool to synthesise VHDL code, execute FPGA place and route and generate a bitstream file to programming the target Xilinx FPGA (also shown in Fig. 3).

In the VRA-based evolvable system, all operations are controlled by the control and interface, which communicates with the onboard SRAM and responses to the host PC program. The EA unit implements the evolutionary operations and generates configuration bit string (chromosome) to configure the FE array unit. The FE array unit processes the system inputs set and its function is reconfigurable. The fitness unit calculates individual fitness by comparing the outputs set from the FE array unit with the predefined expected outputs set. Once the system evolution is finished, the back end signals the host PC program the completion of the evolutionary operation. The results, which are first stored in the onboard SRAM, is transmitted to the host PC and saved as a result file.

4.1 Gate level evolution of character recognition systems

First, the proposed VRA is devoted to evolve a character recognition system using gate level evolution. As an alternative to conventional artificial neural network (ANN) approaches to pattern recognition, EHW has been employed for the evolutionary design of high-speed pattern recognition system in recent years [8–10, 34]. The significant benefits of the EHW approach to pattern recognition compared with ANN are the high-processing speed of the hardware and the readability of the learned result, which have been discussed in the literature [8]. It is worth pointing out that the work presented herein mainly focuses on demonstrating the possibility and efficiency of applying VRA-based evolvable system for the gate level evolution of pattern recognition system, rather than evolving a complicated character recognition system which have been successfully implemented in our previous works with different incremental evolution strategies [35, 36].

Fig. 4 gives an outline of the target evolvable character recognition system, which is expected to identify 16 different binary patterns (characters from A to P). To process the input characters of 5×6 pixels, where each pixel can be 0 or 1, the character recognition system includes a 30 bit input port. Each 1 bit input responds to one pixel in an input character. As shown in Fig. 4, each single bit output port of the evolvable system corresponds to one character in the training characters set. Therefore the proposed system consists of 16 single bit output ports. During the characters-distinguishing process, the output port of the evolved system corresponding to the input character should be 1; synchronously the other 15 single bit outputs should be 0. According to this feature, 16 different

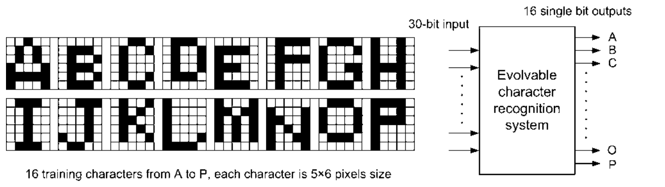
30-bit input
16 single bit outputs
Evolvable character recognition system
A
B
C
O
P
16 training characters from A to P, each character is 5×6 pixels size

Figure 4 *Diagram of evolvable character recognition system for patterns A to P identification*

characters in the training characters set can be recognised individually.

4.1.1 FE array unit:

As discussed in [37], the size of the FE array has significant effects to the system performance. To achieve a reasonable hardware cost and system performance, in this work, the size of the FE array is selected by a parameter-tuning process. Different topologies of the FE array were tested in our prior experiments to find an optimised size of the FE array. For the proposed character recognition system, which includes 30 bit input and 16 single bit outputs, we decide to employ an FE array, which consists of four FE columns. Sixteen uniform two-input FEs are placed in each column. Each FE's two inputs in column l ($l = 2, 3, 4$) can be connected to any one output of FEs in column $l - 1$. As shown in Fig. 5, the input connections of each FE are selected by its two equipped 16-to-1 multiplexers. In column 1, each FE employs two 32-to-1 multiplexers. Therefore each input of FE in column 1 can be connected to any one bit of the 30 bit system input or defined as a bias of value 1 or 0. In the FE array, each FE in the last column corresponds to one system output. Each FE in columns 2, 3, 4 can be programmed to perform one of eight functions that can be seen in Fig. 5. Only two functions of 'a' and 'not b' are available for FEs in column 1. Each FE is equipped with a Flip-flop to support pipeline processing. A column of FEs is considered as a single stage of the pipeline. Each FE needs 11 bit (5 + 5 + 1 bits in column 1, 4 + 4 + 3 bits in other columns) to determine its input connections and function. In total, the chromosome which is stored in a configuration memory is $4 \times (11 \times 16) = 704$ bits. In our approach, 16 FEs in the same column are configured simultaneously and we need four system clocks to completely change the configuration of the FE array. Given these features, the 704 bit chromosome in the configuration memory is divided and stored in four configuration banks (cnfBank) of 176 bits.

4.1.2 Fitness unit:

The fitness unit evaluates the fitness of candidate circuits uploaded to the FE array unit by reading its output vectors and comparing them against the expected output vectors. The fitness function is calculated as follows

$$\text{Fitness} = \sum_{\text{vector}} \sum_{\text{output}} x; \quad \text{where } x = \begin{cases} 1 & \text{output} = \text{expect} \\ 0 & \text{output} \neq \text{expect} \end{cases} \tag{1}$$

For an output vector, each single bit output is compared with its corresponding expected system output (which is labelled as expect). If they are equal, the variable x will be presented as 1 and be added to the fitness function by using an accumulator. Fitness is the sum values for the compared results of all 16 single bit outputs (output) in the processed 16 output vectors (vector) set. The maximum fitness value is $16 \times 16 = 256$.

4.1.3 EA unit and control and interface:

Fig. 6 shows the hardware implementation of the EA unit. The EA unit consists of a $4 \times 4 \times 176$ bit population memory (PM), a 4×176 bit best chromosome memory (BCM), a 9 bit fitness memory (FM), a random number generator (RNG) and an MO.

PM, BCM and FM: Because the FE array has to be configured in four clocks per 176 bit, the organisation of the PM is $4 \times 4 \times 176$ bits (corresponding to 4 individuals) and the BCM is 4×176 bits. The FM is devoted to store the fitness value of the best chromosome and is 9 bits. All are realised using flip-flops available in the FPGA.

RNG: The RNG is used in two steps of evolution: (1) randomly generating the initial population; and (2) selecting the mutation positions of a chromosome. The RNG generates a sequence of pseudorandom binary bit strings based on the theory of linear cellular automata (CA) [38]. The CA used in the RNG consists of 176 altering

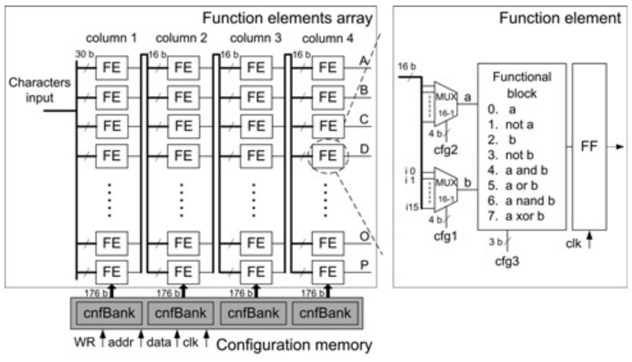

Figure 5 *FE array unit for the evolution of character recognition system*

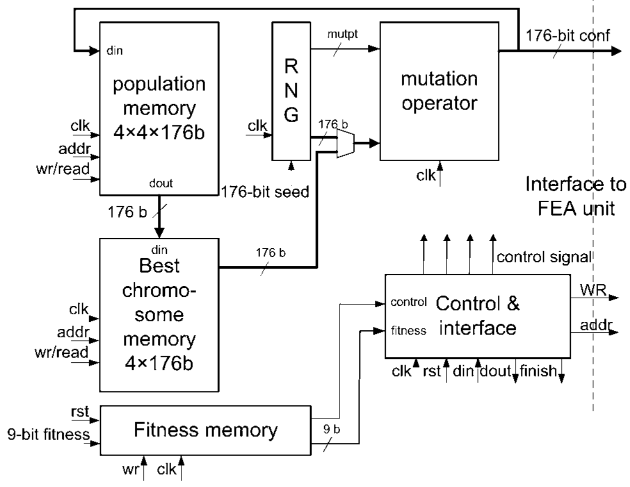

Figure 6 *EA unit and control and interface*

cells, which can change their state according to rules 90 and 150 as detailed in the literature [39].

MO: The MO processes each 4 × 176 bit chromosome in four system clocks. In each clock, the mutation operator reads the mutation position parameters (mutpt) from RNG and inverts a given number (this number is configurable) of randomly selected bits per 176 bits.

The EA unit works as follows: first, EA unit receives a start signal from control and interface. Then the PM is filled by four individuals, which are the mutated versions of four 4 × 176 bit pseudorandom binary numbers generated by the RNG. To obtain the fitness value of each individual, each of them is downloaded to the FE array unit to create a candidate circuit. By comparing the candidate circuit's output set with the expected output set in the fitness unit, the fitness value of each individual can be obtained. After that, the individual with the best fitness in the initial population is selected and the new population is generated by using the fittest individual and four mutants. This is repeated until one of the stop criteria of EA is satisfied. For the application of evolving a character recognition system, the stop criteria of EA are defined as: (1) the predefined generations (2^{25}) are exhausted; or (2) EA finds the expected character recognition circuit.

The control and interface is realised by means of the finite state machine and four address counters (for the write/read operations of PM and BCM in the EA unit), which plays the role of communicating with outside environments (host PC and onboard SRAM) and controlling other units in the evolvable system. When the evolution begins, the control and interface receives a start signal from the host PC, reads data from onboard SRAM and forwards it to the input of the FE array unit. The first valid FE array unit output vector is generated in four FPGA clocks (because of four columns of FEs). However, thanks to pipelined processing, after the initial stage, the FE array unit can generate one output vector per a clock. Hence, 16 input characters can be processed in $15 + 4 = 19$ FPGA clocks. The operations of the remaining components of the evolvable system are also maintained by the control and interface. In the proposed VRA, all the EA operation time, the FE array reconfiguration time and the fitness calculation time are designed to be hidden in the period of processing system input vectors totally. When the stop criteria of the EA is satisfied, control and interface writes result data to onboard SRAM and sends a stop signal to host PC to stop evolution.

4.2 Function level evolution of image filters

To illustrate the flexibility and the power of the proposed VRA, our second example is a spatial image filter, which is evolved at function level. Generally, image noises can be processed either in the frequency domain or in the spatial domain. The spatial image filter is preferable for the EHW-based implementations, as it is computationally cheaper than the frequency image filter. Different intrinsic EHW-based approaches [6, 11] have been proposed for the evolutionary design of spatial image filters to deal with different additive noises, such as Gaussian, uniform and salt-and-pepper noises. In this work, we extend the approach initially developed by Wang *et al.* [12] who have successfully evolved several spatial image operators by using VRA-based EHW.

In our approach, each spatial image filter realised in the FE array unit will be considered as a digital circuit of nine 8 bit inputs (labelled in Figs. 7 and 8 as I0-I8) and a single 8 bit output, which processes grey scale images of 256 × 256 pixels (8 bit/pixel). Any one pixel of filtered image is generated by using a 3 × 3 neighbourhood window. The graphical representation of neighbourhood window operation is shown in Fig. 7, wherein every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbours in the input image.

In contrast to the evolution of the character recognition system in which gate-level-based functions (OR, AND, XOR etc.) and 1 bit connection wires are utilised, we approach the problem of evolvable image filter using functional level evolution where functional operations (maximum, minimum, average etc.) and 8 bit datapaths are employed in the FE array. As shown in Fig. 8, the FE array consists of seven FE columns from system input to output. Except for the last column, eight uniform FEs are placed in each column. The last column includes only one FE. Every 8 bit input of FEs in column l ($l = 2, 3, 4, 5, 6, 7$)

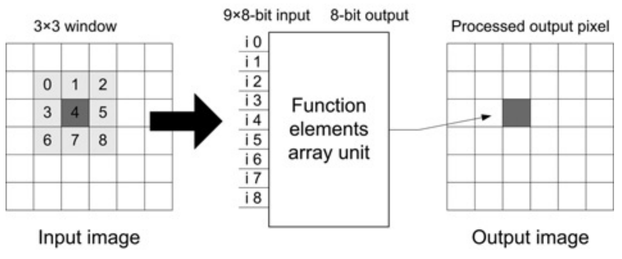

Figure 7 *Neighbourhood window operation for image processing*

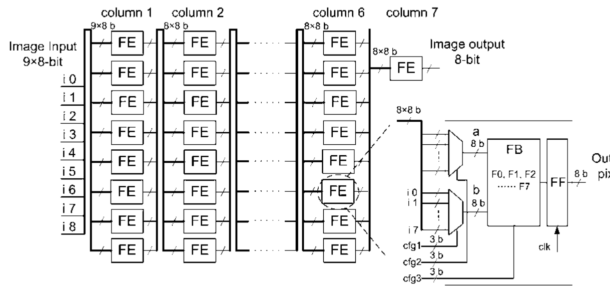

Figure 8 *FE array for the evolution of image filters*

Table 2 Functions implemented in FEs

Index	Function	Index	Function
0	a	4	Min(a,b)
1	$(a+b) \gg 1$	5	$a \ll 1$
2	$(a+b+1) \gg 1$	6	a xor b
3	Max(a, b)	7	b

All functions have 8 bit operands and 8 bit outputs.

can be connected to anyone 8 bit output of FEs in column $l-1$. In column 1, the first input of an FE is constrained to be connected with system inputs I0-I7 and the second input links system input I1-I8. Any FE can be programmed to execute one of eight functions given in Table 2, which is applied to its two 8 bit input a and b. The selected eight functions have been proved to be efficient for the application of evolvable image filter in our previous experiments [12]. A single FE is encoded in the chromosome by using 9 bit genes, 3 + 3 bits (cfg 1 and cfg 2) are used to control the selections of FE's inputs, 3 bits (cfg 3) are used to select the implemented function of an FE. For the FE array consisting of 49 elements, the chromosome length is $(6 \times 8 \times 9) + 9 = 441$ bits.

EA unit needs to process 72 bit configuration bit string per clock for the configuration of one FE column. Therefore the EA unit employs a $4 \times 7 \times 72$ bit PM, a 7×72 bit BCM and a 24 bit FM. The datapaths of RNG and MO are also revised for processing 72 bit data.

The single evolutionary objective in our experiment is to minimise the difference between the filtered image and the original uncorrupted image. To measure the quality of the filtered image, the mean difference per pixel (MDPP)-based fitness function is implemented in the fitness unit. The original image size is $k \times k$ ($k = 256$), but only a sub-area image of $(k-2) \times (k-2)$ pixels is considered, because the pixels at the image borders are untouchable for the 3×3 window and remain unfiltered. The MDPP-based fitness function is described as follows

$$\text{fitness}_{\text{MDPP}} = \sum_{i=1}^{k-2} \sum_{j=1}^{k-2} \left| v(i,j) - w(i,j) \right| \qquad (2)$$

The fitness calculation is executed by the fitness unit. MDPP-based fitness function is calculated by comparing the diversity of each pixel in the filtered image v with its corresponding pixels w in the original uncorrupted image.

5 Experimental results

In this section, we present and summarise the experimental results of evolving character recognition systems and image filters. In Table 3, the parameters of employed EAs are given. Different mutation rate settings were tested in our experiments to explore the effects of the diversity of EA parameters for different applications.

Table 3 EA parameters and features

Parameters	Evolvable character recognition system	Evolvable image filter
size of FE array	4 columns, 16 rows	7 columns, 8 rows
chromosome length, bits	704	441
population size($1+\lambda$)	$1+4$	$1+4$
selection	elitism	elitism
genetic operations	bit-flip mutation	bit-flip mutation
mutation rate, %	0.1, 0.2, 0.4, 0.8	0.8, 1.6, 3.2, 6.4
termination criteria for evolutionary process	(1) the predefined 2^{25} generations are exhausted; or (2) EA finds the expected solution	the predefined 2^{14} generations are exhausted

5.1 Synthesis for Celoxica RC1000 PCI board

Both of the evolvable character recognition system and the evolvable image filter were described in VHDL. After simulations with Modelsim, the designs were synthesised using Xilinx ISE 6.3 to Virtex xcv2000E FPGA, which was available on the Celoxica RC1000 PCI board. Table 4 summarises the synthesis results for the evolvable character recognition system and the evolvable image filter.

According to the synthesis reports, the maximum FPGA clock frequencies attained are 97.513 MHz for the

Table 4 Synthesis resluts in Virtex xcv2000E FPGA

Resource	Available	Used for character recognition system	Used for image filter
slices	19 200	5617 (29%)	6711 (35%)
slice flip flops	38 400	6014 (16%)	5055 (13%)
four input LUTs	38 400	6183 (16%)	10 132 (26%)

evolvable character recognition system, and 79.233 MHz for the evolvable image filter. However, the real running speed on the Celoxica RC1000 PCI board for all of the experiments was limited to 33 MHz because of easier synchronisation with the current version of the PCI interface employed by the prototyping board, which can operate correctly with PCI bus clocks from 0 to 33 MHz.

5.2 Evolution speed

As the pipeline process is supported by the proposed VRA, all the EA operations time, the fitness value calculation time and the reconfiguration time of FE array could be overlapped by the period of processing system input vectors set. Therefore if the size of input vectors set is v, and the operation frequency of the hardware platform is f MHz, it is possible to express the fitness evaluation time for a single individual t_{eval} as

$$t_{\text{eval}} = \frac{v}{f} \tag{3}$$

The total evolution time t for the proposed evolvable platform can be expressed as

$$t = t_{\text{init}} + \text{ngen} \times p \times t_{\text{eval}} = t_{\text{init}} + \text{ngen} \times p \times \frac{v}{f} \tag{4}$$

where ngen is the number of generations, p the population size and t_{init} is the time required to generate the first valid output of FE array in the pipeline process (several FPGA clocks only).

We also simulated the same algorithm in software (which was described in C code and compiled by using MS Visual C++ 6.0) on a 2.0 GHz Pentium IV processor with 1.5 Gb of DDR SDRAM for verification and comparison with the FPGA-based implementation described above. The evolution times for the evolution runs of 10 000 generations of four individuals in the two applications: character recognition system and image filter were measured for speed comparison. As can be seen in Table 5, although the clock frequency of the Pentium IV processor is approximately 61 times greater than that of the FPGA, our FPGA-based evolvable platform still gives better results in terms of evolution time. The speed-up factors for evolution time ($\text{Speedup}_{\text{time}}$) obtained by means of FPGA implementation are 169 for evolving character recognition system and 115 for evolving image filter, respectively. Note that we do not compare the total system evolution time here.

5.3 Results of evolving character recognition systems

We performed 100 runs per mutation rate setting, and the initial seed for the initial population in each EA run was generated at random. All 100 EA runs were successful in all cases. One successful run means EA can find a feasible result circuit within predefined number of generations. Table 6 summarises the experiments. We can find that the evolvable character system has the best performance when the mutation rate 0.2% is chosen. All the average and stand deviation (std. dev.) results in Table 6 were calculated for 100 EA runs. An example of the evolved character recognition system is shown in Fig. 9, which employs 41 FEs. All predefined functions (e.g. NOT, AND, OR, NAND, XOR) which are available in the functional block of FE (see Fig. 5) appear in the result circuit. It proves that these selected functions are useful for our task.

Table 5 Evolution time comparison of the FPGA implementation with the Pentium IV PC implementation

Application	Platform	Clock frequency, MHz	Evolution time, s	$\text{Speed-up}_{\text{time}}$
evolvable character recognition system	FPGA	33	0.019	169
	Pentium IV	2000	3.204	
evolvable image filter	FPGA	33	78.201	115
	Pentium IV	2000	8988.725	

Table 6 Results for evolving character recognition systems by using different mutation rate settings

Mutation rate, %	Total evolution time, s (avg.)	Number of generations			
		avg.	std. dev.	min	max
0.1	2.326	1 199 599	1 709 702	145 841	13 597 095
0.2	1.108	571 361	556 557	79 494	2 978 566
0.4	4.246	2 189 172	2 331 161	141 678	12 317 617
0.8	3.483	1 795 993	1 348 949	288 986	7 454 084

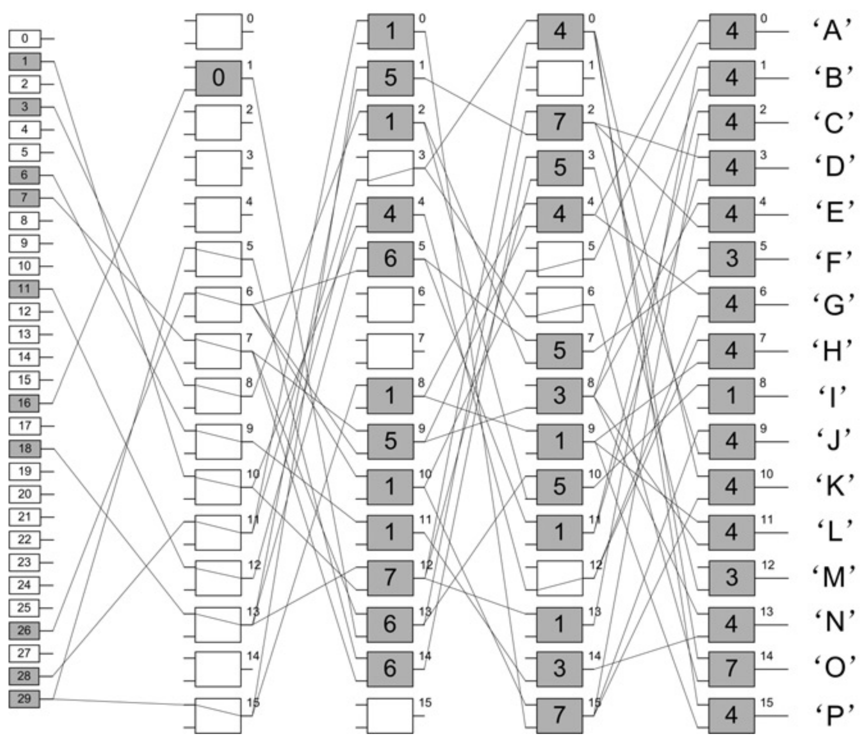

Figure 9 *Evolved character recognition system on the FPGA*

The functions of FEs are numbered according to Fig. 5

5.4 Results of evolving image filters

Our proposed evolvable image filter was tested on processing two kinds of additive noise: Gaussian noise (mean 0 and variance 0.008) and Salt & Pepper noise (the image contains 5% corrupted pixels with white or block shots) which were independent of the image itself. Both of these noises were generated by using Matlab functions. Two types of image filters were evolved to process these noises, respectively. In our experiment, all the evolved image filters were trained by using Lena image, which was a grey scale (8 bits each pixel) image of 256 × 256 pixels.

We performed 100 independent runs for each experimental setting and measured the best and average MDPP by comparing the filtered images with the original uncorrupted image. As the MDPP function measures the diversities between filtered and original uncorrupted image, the lower the MDPP value the better is the quality of the filtered image. Each evolved result was achieved after 16 384 generations of EA run. Table 7 summarises the evolved filters obtained for processing Salt & Pepper noises and Gaussian noise under different EA parameter settings. According to our results, the mutation rate of 3.2% seems a reasonable parameter setting under this experimental frame.

We can compare the obtained Salt & Pepper filters with the results reported in other literature. In [6], Martínek and Sekanina employed a CGP inspired evolvable image filter to process the same 5% Salt & Pepper noise. Their obtained average MDPP values for Salt & Pepper noise are 3.75, 3.06, 4.11 and 3.69 for different EA settings. Table 7 shows that the performance of our evolved Salt & Pepper noise filters outperforms the results of [6] in the item of average MDPP. On the other hand, it is worth mentioning that the best value of MDPP for Salt & Pepper noise obtained in their approach is better than that in our approach (MDPP 0.48 against 1.54). However, average MDPP value is a more crucial factor than best MDPP value to evaluate the performance of evolvable filters

Table 7 Results for evolving Salt & Pepper noise (5% corrupted pixels with white or black shots) filters and Gaussion noise (mean 0 and variance 0.008) filters

Type of noise	Mutation rate, %	Best MDPP	Average MDPP	Total evolution time, s
Salt & Pepper	0.8	1.80	3.15	128.125
	1.6	1.62	2.43	
	3.2	1.54	2.30	
	6.4	1.86	2.68	
Gaussian	0.8	8.48	9.18	
	1.6	8.39	8.94	
	3.2	8.65	9.06	
	6.4	8.73	9.34	

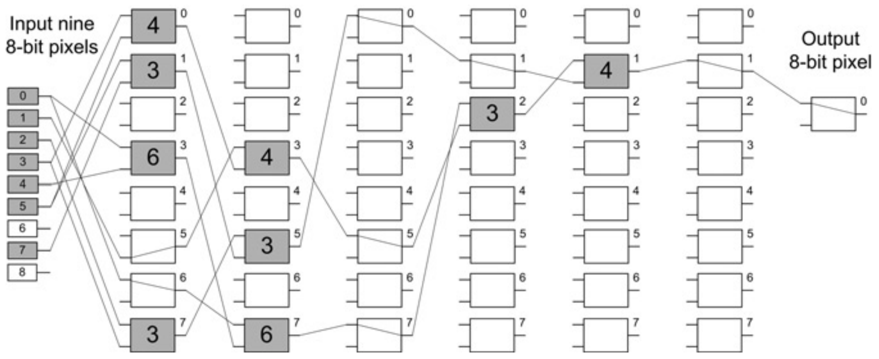

Figure 10 *Example of evolved Salt & Pepper filter, which was trained by Lena image*
The functions of FEs are numbered according to Table 2

because it gives an example of the result which we can obtain with the highest probability at the end of EA run.

An example of the evolved image operator for processing Salt & Pepper noise is shown in Fig. 10. It consists of nine FEs and employs three functions: F3, F4 and F6. Fig. 11 shows an evolved Gaussian noise filter, which consists of eight FEs and utilises only two functions: F1 and F2. According to the obtained results, it seems that our evolvable system tends to employ different function sets to process the existed Salt & Pepper noise and Gaussian noise, respectively. After the image filters were evolved, baboon image and cameraman image with the same Salt & Pepper noise or Gaussian noise were employed as test images to test the generality of the evolved image filters, which were trained by the Lena image. Figs. 12 and 13 show the filtered images, which are processed by the evolved Salt & Pepper noise filter and Gaussian filter shown in Figs. 10 and 11, respectively.

6 Discussion

Our proposed VRA is described using VHDL. Benefiting from this generic HDL level-based approach, VRA can be easily modified and developed for implementing different evolvable systems in a very short time. By changing the parameters of some functional units in the VRA (such as the size of the chromosome memory, the employed functions set and datapaths in each FE, the size and topology of the FE array etc.), both of gate level-based evolvable character recognition system and function level-based image filter were performed successfully in our VRA-based evolvable platform.

Real-time adaptation ability to a different environment is a key feature expected for EHW. Our proposed evolvable platform seems to be a promise for this purpose, as an acceptable result circuit for a certain environment can be evolved in a reasonable evolution time (several seconds for some applications). On the other hand, since all of the components (i.e. FE array, EA, fitness calculation etc.) of our proposed evolvable system are implemented on a single FPGA, this approach is potentially suitable for the applications of adaptive embedded system wherein a common intrinsic EHW, which generally employs a PC to execute EA, cannot be used.

Since one of the main aspects of the proposed VRA-based approach in this paper is its capability to improve the performance related to computational efforts, we decided to do a comparison of computational costs for our FPGA-based hardware implementation and the PC-based software simulation. This comparison is presented in Table 5. In both of evolving character recognition system and evolving

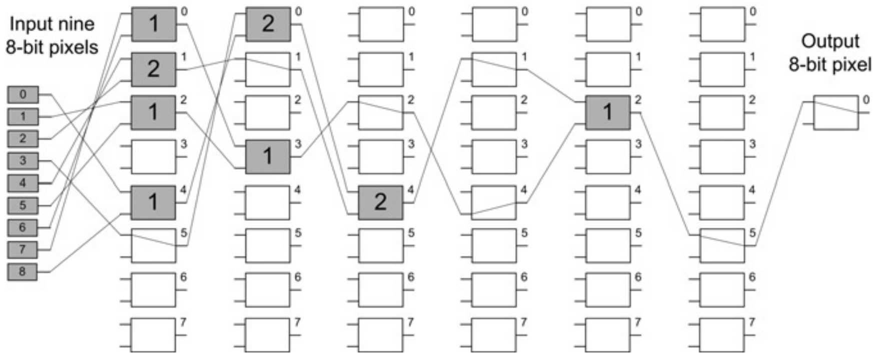

Figure 11 *Example of evolved Gaussian filter, which was trained by Lena image*
The functions of FEs are numbered according to Table 2

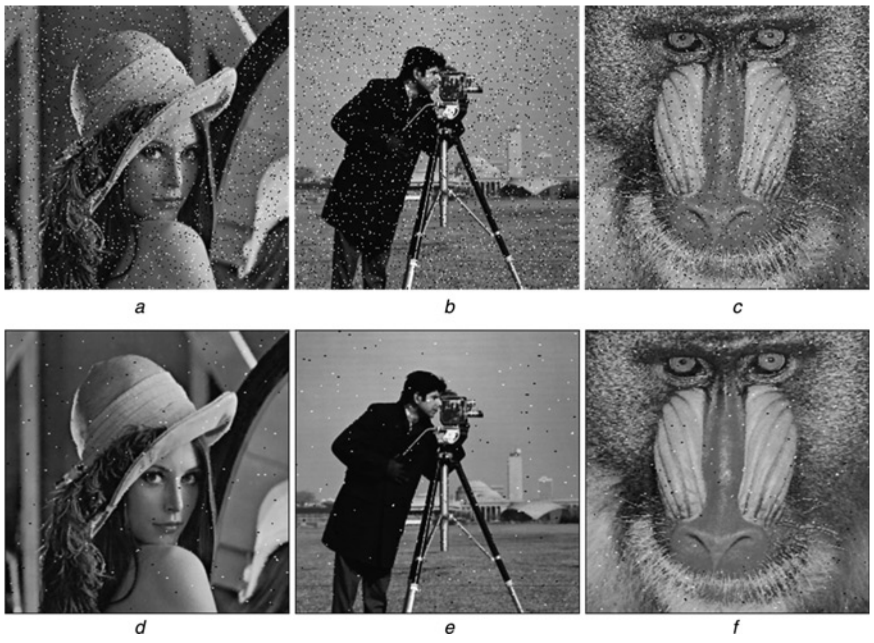

Figure 12 *Examples of images filtered by the evolved Salt & Pepper filter as shown in Fig. 10*

a Corrupted Lena image
b Corrupted cameraman image
c Corrupted baboon image
d Filtered Lena image (MDPP = 1.86)
e Filtered cameraman image (MDPP = 2.00)
f Filtered baboon image (MDPP = 5.96)

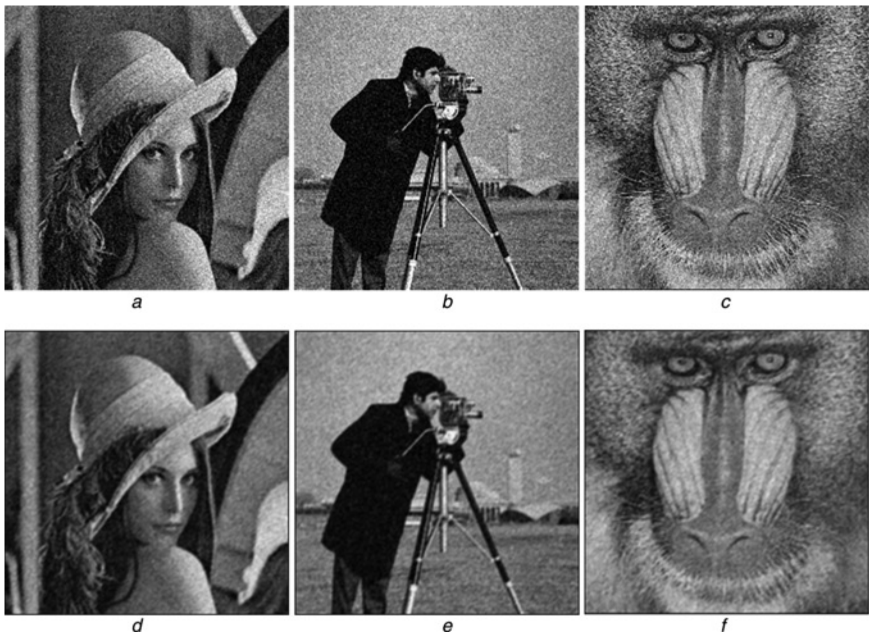

Figure 13 *Examples of images filtered by the evolved Gaussian filter as shown in Fig. 11*

a Corrupted Lena image
b Corrupted cameraman image
c Corrupted baboon image
d Filtered Lena image (MDPP = 8.39)
e Filtered cameraman image (MDPP = 9.73)
f Filtered baboon image (MDPP = 17.35)

image filter, the proposed FPGA implementation (Virtex xcv2000E FPGA/33 MHz) yields a significant speedup of two orders of magnitudes over software implementation (Pentium IV/2 GHz). We believe that the performance advantage obtained with the proposed scheme is mainly because of the features of pipeline, parallelisation and no function call overhead in the VRA. Pipeline technology is common in hardware design to multiple, concurrently executing available resources on the chip. In our hardware implementation, the task of processing system input vectors set in the FE array is completely pipelined. A column of FEs is considered as a single stage of the pipeline, and thus the FE array is able to process each input vector in one FPGA clock. As pointed out by Glette *et al.* [27], there is a large overhead associated with fitness calculation in software simulation. However, in our hardware implementation, the EA and fitness calculations incur no overhead for them because of the uses of optimised state machines and custom address counter. These operations can be executed simultaneously with the FE array-pipelining process and thus the entire evolvable system can be pipelined. Pipeline strategy can be considered as a kind of parallelism: fine-grain parallelism [31]. To bring more potential performance improvement, it is possible to implement other kinds of parallelism: coarse-grain parallelism in our VRA. For instance, we have successfully implemented three VRA-based evolvable cores on a single FPGA to execute incremental evolutions of combinational logic circuits in parallel, which offers significantly higher performance than single VRA core-based implementation [40].

Another interesting aspect of this work is the analysis of the time required for the individual fitness evaluation in FPGA configuration bitstream-based intrinsic EHW and our proposed virtual reconfiguration technology-based intrinsic approach. In traditional FPGA configuration bitstream-based intrinsic EHW, the evaluation of each individual requires several steps: (1) reconfigure FPGA; (2) process input vectors set on FPGA; (3) transfer output data to processor and calculate the individual fitness value. As discussed in Section 2.1, the FPGA reconfiguration time is generally kept in the interval of 1–100 ms in most of partial reconfiguration-technique-based applications. Limited by this theoretical boundaries of FPGA devices, we can estimate that the minimum fitness evaluation time in traditional FPGA configuration bitstream-based intrinsic EHW should be more than 1 ms, even though the input vectors set processing time on FPGA and the fitness calculation time on processor are not taken into account in this estimation. On the other hand, as mentioned in Section 5.2, the fitness evaluation time in the VRA-based intrinsic EHW is depended only on the input vectors set processing time. The time required for system reconfiguration can be ignored in the VRA. This feature of virtual reconfiguration technology can bring us a significant performance improvement in terms of fitness evaluation time. For instance, in [18], Hollongworth *et al.* proposed a JBits-technique-based partial reconfigurable platform to evolve a simple 2 bit adder with 16 input vectors. Limited to their old and fairly slow interface, the ISA bus at 8 MHz, ~22.8 ms, was required to perform one individual fitness evaluation. In theory the current Virtex device can be programmed at up to 66 MHz, bringing the average fitness evaluation time down to 2.8 ms. However, this time is still longer than our proposed virtual reconfigurable-technique-based approach. In our evolvable platform wherein the FPGA operates at 33 MHz, for the same application of evolving 2 bit adder, the single individual fitness evaluation time is decreased to ~0.5 μs.

In [7], Sekanina has mentioned that the main disadvantage of the virtual reconfiguration technique was the high implementation cost if compared with the size of its evolved circuits. For realising the feature of virtual reconfiguration, much of the FPGA resources are wasted on the chromosome memory and the FE's inputs routing multiplexer implementations. In our proposed scheme, we try to approach this scalability issue by using a new two-dimensional geometric structure-based phenotype representation to replace CGP-based representation employed by Sekanina. We can compare the implementation cost of our approach with their CGP-inspired scheme [6] in the application of evolvable image filter. Although the employed population size and the size of FE array utilised in our approach are similar to their approach (four individuals, eight columns and seven rows), the CLB slices cost in our evolvable platform (6711 slices) is only 67% of Sekanina's implementation (10 042 slices). This is mainly because of the fact that the smaller multiplexer and chromosome memory are used in our approach by introducing more restrictions in our FE array. For example, in Sekanina's approach, to allow that the inputs of each FEs can be connected to the primary system inputs, each input of FE is routed by a 16-to-1 multiplexer and each FE needs $4 + 4 + 3$ bit genes to decide its inputs connection and implemented function. However, in our implementation, the size of routing multiplexer is 8-to-1 and only $3 + 3 + 3$ bit genes are required for each FE. Although our proposed phenotype representation can efficiently reduce the implementation cost by introducing more restrictions in FE array, a slight performance penalty of the method appears in our experiments. For instance, the experimental results presented in Section 5.3 have shown that the quality of our best evolved Salt & Pepper noise filter was poorer than Sekanina's best result, even though the average quality of our evolved circuits was high, comparable with Sekanina's approach [6]. This observation implies that our proposed scheme loses some capabilities of obtaining innovative and high-quality solutions, because more constrains are included in the interconnections of our FE array than CGP-based representation.

7 Conclusions

In this paper, we have presented a virtual reconfiguration-technique-based intrinsic evolvable platform, known as

VRA. This evolvable platform is described in VHDL and all of the components of evolvable system including FE array, EA unit, and fitness calculation unit can be realised on a Xilinx Virtex xcv2000E FPGA to achieve the feature of complete hardware implementation. The proposed architecture is generic and can efficiently and flexibly implement different gate level or functional level-based evolvable systems by modifying its HDL description. The prototype of the VRA-based evolvable system was tested by evolving two different real-world applications: the character recognition system and the spatial image filter. The powerful computation ability of our proposed evolvable platform is presented in the experimental results. When compared with the equivalent software simulation, our FPGA implementation obtains a performance increase of over 100 times in all cases. On the other hand, according to the analysis of fitness evaluation process in intrinsic evolution, the proposed VRA shows a promise to avoid the bottleneck introduced by the slow reconfiguration speed in traditional FPGA configuration bitstream-based intrinsic EHW. Future work will be concentrated on developing the reported VRA for solving more complex and demanding real-world industrial problems.

8 Acknowledgment

This work was supported by INHA University Research Grant.

9 References

[1] GREENWOOD G.W., TYRRELL A.M.: 'Introduction to evolvable hardware: a practical guide for designing self-adaptive systems' (Wiley-IEEE Press, 2006, 1st edn.)

[2] ZEBULUM R.S., STOICA A., KEYMEULEN D., SEKANINA L.: 'Evolvable hardware system at extreme low temperatures'. Proc. 6th Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Sitges, Spain, September 2005, pp. 37–45

[3] SEKANINA L., STARECEK L., GAJDA Z., KOTASEK Z.: 'Evolution of multifunctional combinational modules controlled by the power supply voltage'. Proc. 1st NASA/ESA Conf. Adaptive Hardware and Systems (AHS 2006), Istanbul, Turkey, June 2006, pp. 186–193

[4] MILLER J.F., THOMSON P.: 'Cartesian genetic programming'. Proc. 3rd European Conf. Genetic Programming, Edinburgh, Scotland, UK, April 2000, pp. 121–132

[5] SEKANINA L., FRIEDL S.: 'On routine implementation of virtual evolvable devices using COMBO6'. Proc. 2004 NASA/DoD Conf. Evolvable Hardware, Seattle, USA, June 2004, pp. 63–70

[6] MARTÍNEK T., SEKANINA L.: 'An evolvable image filter: experimental evaluation of a complete hardware implementation in FPGA'. Proc. 6th Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Sitges, Spain, September 2005, pp. 76–85

[7] SEKANINA L.: 'Virtual reconfigurable circuits for real-world applications of evolvable hardware'. Proc. 5th Int. Conf. Evolvable Systems: From Biology to Hard-ware (ICES), Trondheim, Norway, March 2003, pp. 186–197

[8] IWATA M., KAJITANI I., YAMADA H., IBA H., HIGUCHI T.: 'A pattern recognition system using evolvable hardware'. Proc. Int. Conf. Parallel Problem Solving from Nature IV (PPSN IV), Berlin, German, September 1996, pp. 761–770

[9] TORRESEN J.: 'A scalable approach to evolvable hardware', *Genet. Program. Evol. Mach.*, 2002, **3**, (3), pp. 259–282

[10] WANG J., PIAO C.H., LEE C.H.: 'FPGA implementation of evolvable characters recognizers with self-adaptive mutation rates'. Proc. 2007 Int. Conf. Adaptive and Natural Computing Algorithms (ICANNGA), Warsaw, Poland, April 2007, pp. 286–295

[11] ZHANG Y., SMITH S.L., TYRRELL A.M.: 'Digital circuit design using intrinsic evolvable hardware'. Proc. 2004 NASA/DoD Conf. the evolvable Hardware, Seattle, USA, June 2004, pp. 55–62

[12] WANG J., JUNG J.K., LEE C.H.: 'Evolutionary design of image filter using the Celoxica RC1000 board'. Int. Conf. Control, Automation and Systems (ICCAS), Gyeong Gi, Kintex, Korea, June 2005, pp. 1355–1360

[13] SMITH S., LEGGETT S., TYRRELL A.M.: 'An implicit context representation for evolving image processing filters'. Proc. 2005 EvoWorkshops: EvoBIO, EvoCOMNET, EvoHot, EvoIASP, EvoMUSART, and EvoSTOC, Lausanne, Switzerland, March 2005, pp. 407–416

[14] XILINX.: 'Xc6200 FPGA Data Sheet', 1997

[15] THOMPSON A., LAYZELL P., ZEBULUM S.: 'Explorations in design space: unconventional electronics design through artificial evolution', *IEEE Trans. Evol. Comput.*, 1999, **3**, (3), pp. 167–196

[16] XILINX INC.: 'Virtex field programmable gate arrays databook', 1999

[17] XILINX.: 'JBits 2.0.1 Documentation', 1999

[18] HOLLINGWORTH G., SMITH S., TYRRELL A.M.: 'The intrinsic evolution of Virtex devices through internet reconfigurable logic'. Proc. 3rd Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Edinburgh, Scotland, UK, April 2000, pp. 72–79

[19] TYRRELL A.M., KROHLING R.A., ZHOU Y.: 'Evolutionary algorithm for the promotion of evolvable hardware', *IEE Proc., Comput. Digit. Tech.*, 2004, **151**, (4), pp. 267–275

[20] UPEGU A., SANCHEZ E.: 'Evolving hardware by dynamically reconfiguring Xilinx FPGAs'. Proc. 6th Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Sitges, Spain, September 2005, pp. 56–65

[21] UPEGU A., SANCHEZ E.: 'Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs'. Proc. 1st NASA/ESA Conf. Adaptive Hardware and Systems (AHS), Istanbul, Turkey, June 2006, pp. 153–160

[22] ANTONI L., LEVEUGLE R., FEHER B.: 'Using run-time reconfiguration for fault injection in hardware prototypes'. Proc. IEEE Int. Symp. Defect and Fault Tolerance in VLSI Systems, Yamanashi, Japan, October 2000, pp. 405–413

[23] PORTER R., MCCABE K., BERGMANN N.: 'An application approach to evolvable hardware'. Proc. 1st NASA/DoD Workshop on Evolvable Hardware, Pasadena, California, USA, July 1999, pp. 170–174

[24] SLOARCH C., SHARMAN K.: 'The design and implementation of custom architectures for evolvable hardware using off-the-shelf progarmmable devices'. Proc. 3rd Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Edinburgh, Scotland, UK, April 2000, pp. 197–207

[25] HADDOW P., TUFTE G.: 'Bridging the genotype–phenotype mapping for digital FPGAs'. Proc. 3rd NASA/DoD Workshop on Evolvable Hardware, Long Beach, California, USA, July 2001, pp. 109–115

[26] GLETTE K., TORRESEN J.: 'A flexible on-chip evolution system implemented on a Xilinx Virtex-II Pro Device'. Proc. 6th Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Sitges, Spain, September 2005, pp. 66–75

[27] GLETTE K., TORRESEN J., YASUNAGA M., YAMAGUCHI Y.: 'On-chip evolution using a soft processor core applied to image recognition'. Proc. 1st NASA/ESA Conf. Adaptive Hardware and Systems (AHS), Istanbul, Turkey, June 2006, pp. 373–380

[28] COELLO COELLO C.A., CHRISTIANSEN A.D., AGUIRRE A.H.: 'Towards automated evolutionary design of combinational circuits', *Comput. Electr. Eng.*, 2001, **27**, (1), pp. 1–28

[29] SEKANINA L., FRIEDL S.: 'An evolvable combinational unit for FPGAs', *Comput. Inf. (Slovakia)*, 2004, **23**, (5), pp. 461–486

[30] MILLER J.F., JOB D., VASSILEV V.K.: 'Principles in the evolutionary design of digital circuits – Part I', *Genet. Program. Evol. Mach.*, 2000, **1**, (1), pp. 8–35

[31] MARTIN P.: 'A hardware implementation of a genetic programming system using FPGAs and Handel-C', *Genet. Program. Evol. Mach.*, 2001, **2**, (4), pp. 317–343

[32] BENSAALI F., AMIRA A., BOURIDANE A.: 'Accelerating matrix product on reconfigurable hardware for image processing applications', *IEE Proc., Circuits, Devices Syst.*, 2005, **152**, (3), pp. 236–246

[33] EMBEDDED SOLUTIONS LTD.: 'Handel-C language reference manual V2.3', 2001

[34] YASUNAGA M., NAKAMURA T., YOSHIHARA I., KIM J.H.: 'Genetic algorithm-based design methodology for pattern recognition hardware'. Proc. 3rd Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Edinburgh, Scotland, UK, April 2000, pp. 264–273

[35] WANG J., JUNG J.K., LEE Y.M., LEE C.H.: 'Using reconfigurable architecture-based intrinsic incremental evolution to evolve a character classification system'. Proc. Int. Conf. Computational Intelligence and Security (CIS), Xi'an, China, December 2005, pp. 216–223

[36] WANG J., LEE C.H.: 'Introducing partitioning training set strategy to intrinsic incremental evolution'. Proc. Mexican Int. Conf. Artifical Intelligence (MICAI), Apizaco, Mexico, November 2006, pp. 272–282

[37] KALGANOVA T., MILLER J.F., FOGARTY T.C.: 'Some aspects of an evolvable hardware approach for multiple-valued combinational circuit design'. Proc. 2nd Int. Conf. Evolvable Systems: From Biology to Hardware (ICES), Lausanne, Switzerland, September 1998, pp. 78–89

[38] WOLFRAM S.: 'Universality and complexity in cellular automata', *Physica*, 1984, **10D**, pp. 1–35

[39] SCOTT S.D.: 'HGA: a hardware-based genetic algorithm', Master Thesis, University of Nebraska., 1994 Lincoln

[40] WANG J., PIAO C.H., LEE C.H.: 'Implementing multi-VRC cores to evolve combinational logic circuits in parallel'. Proc. 7th Int. Conf. Evolvable Systems: From Biology To Hardware (ICES), Wuhan, China, September 2007, pp. 23–34