# Evolutionary Design of Image Filter Using The Celoxica Rc1000 Board

Jin Wang*, Je Kyo Jung**, and Chong Ho Lee***

Department of Information Technology & Telecommunication , Inha University, Incheon, Korea

* (E-mail: wangjin_liips@yahoo.com.cn)

** (E-mail: i_nux@hotmail.com)

***(E-mail: chlee@inha.ac.kr)

**Abstract**: In this paper, we approach the problem of image filter design automation using a kind of intrinsic evolvable hardware architecture. For the purpose of implementing the intrinsic evolution process in a common FPGA chip and evolving a complicated digital circuit system-image filter, the design automation system employs the reconfigurable circuit architecture as the reconfigurable component of the EHW. The reconfigurable circuit architecture is inspired by the Cartesian Genetic Programming and the functional level evolution. To increase the speed of the hardware evolution, the whole evolvable hardware system which consists of evolution algorithm unit, fitness value calculation unit and reconfigurable unit are implemented by a commercial FPGA chip. The Celoxica RC1000 card which is fitted with a Xilinx Virtex xcv2000E FPGA chip is employed as the experiment platform. As the result, we conclude the terms of the synthesis report of the image filter design automation system and hardware evolution speed in the Celoxica RC1000 card. The evolved image filter is also compared with the conventional image filter form the point of filtered image quality.

**Keywords:** intrinsic evolution, image filter, function level evolution, Cartesian genetic programming, reconfigurable circuit

## 1. INTRODUCTION

Evolvable hardware (EHW) technique has attracted increasing attention since the early 1990's with the advent of easily reconfigurable hardware such as FPGA (Field Programmable Gate Array) [1]. Evolvable hardware is a hardware which is built on software-reconfigurable logic device (e.g. PLD (Programmable Logic Device) and FPGA) and whose architecture can be reconfigured by using a genetic learning to adapt to new unknown environment [2].

Generally, from the difference of the application field of the EHW, evolvable hardware can be classified as the applications of evolutionary techniques to digital circuit design and a hardware that is capable of online adaptation through reconfiguring its architecture dynamically and autonomously [3]. In our experiment, we focus on applying the EHW technique as an intelligent designer to replace the human being designer in the field of spatial image filter design. Comparing with the traditional engineering design method, evolutionary design brings us some advantages. It brings a higher quality of solution than conventional human being designer. Evolutionary design doesn't require a priori knowledge of any particular design domain, especially when some domain knowledge is available, it can be used to improve the efficiency of the evolutionary design [1].

In recent years, two main EHW evolution methods which are categorized from the point of fitness value evaluated in software simulation or in physical hardware have been established: Extrinsic and Intrinsic evolution [4]. In stead of downloading the best individual to configure the hardware only one times in the extrinsic evolution process, intrinsic evolution tests each individual that is generated by evolution algorithm machine in hardware and the EHW device is reconfigured the same number of times as the population size in each generation.

Evolving a complicated digital circuit needs to employ a complicated criteria to evaluate fitness value. This fitness value evaluation process is quite time consumed. Therefore, in the image filter design automation process, not only the quality of the result circuit output and implementation cost of the result circuit need to be considered, but also the evolution speed is a serious problem. In our experiment, the method of intrinsic evolution is adopted. Contrasted with the extrinsic evolution, the intrinsic evolution takes the design process closer to the real hardware in which each individual is tested out in hardware. It not only increases the accuracy of the evaluation but also provides the opportunity to explore the underlying technology's physical properties [5]. Due to pipelining and parallelism technique which are efficiently implemented by intrinsic evolution process, evolution speed is anticipated improved one to two orders of magnitude over extrinsic evolution.

Some problems need to be considered in the intrinsic evolutionary design process. The first problem is that designer wistfully needs a technique to quickly and availably modify the function and architecture of the hardware system according to the configuration bit-streams which is generated by evolution algorithm component. Another major problem in the intrinsic evolutionary design field is the problem of scale. Most of researches on EHW have a common problem that the evolved circuit size is small. When the size of the truth table of the evolved circuit increased, the time required to calculate the fitness value of a circuit increased quickly.

Faced the above obstacles, partial reconfiguration technique is introduced [6-7] to solve the quickly reconfiguring device problem, but the Xilinx XC6200 serious chip [8] which supported the requirement of partial reconfiguration quitted the market and JBits [9] is too complicated for most of the user. From our viewpoint, we adopted the reconfigurable circuit architecture which is inspired by the Cartesian Genetic Programming [10] and the functional level evolution [2] as the reconfigurable component of the EHW to solve the two problems. In the image filter design automation field, Sekanina [11] first employs the functional level Cartesian Genetic Programming for the extrinsic evolutionary design. A homologous hardware evolution architecture which is named as Virtual Reconfigurable Circuit [12] by Sekanina is introduced but it is not practically implemented by FPGA chip. Yang Zhang [13] applies the similar reconfigurable architecture as evolution technique to intrinsic evolutionary design a digital image filter circuit, but the configuration bit-streams are generated by a outside genetic algorithm processor . It can be considered as a bottleneck for sufficient speed-up the evolution process.

This paper is structured as follows: in section 2, a high level abstraction concept of the evolutionary image filter design automation is presented. Section 3 describes implement of the image filter design automation system in the hardware. Section 4 describes the hardware evolution process. Section 5 describes the experiment result of the hardware implement, the conclusions and future work is summarized in section 6.

## 2. BASIC CONCEPT OF EVOLUTIONARY IMAGE FILTER DESIGN

### 2.1 Evolvable hardware framework for image filter design

Theoretically, a corrupted image can be filtered either in the frequency or in the spatial domain. In our research, we approach the image filter design problem from the spatial domain. In the traditional image filter design field, generally human designer choose different correlated image filters to suppress different predefined image noises. Faced with the different typical noises, various conventional image filters (e.g., mean, median and Gaussian filters) which employ different kernels for a specific noise were detailed in [11]. Our target is to design an image filter design automation system to substitute the traditional human designer and bring us more efficient and lower hardware device cost image filter than the conventional image filter. A kind of FPGA based intrinsic evolvable hardware architecture is applied to compose the evolutionary design automation system. The reconfigurable unit of the intrinsic evolvable hardware is performed as a reconfigurable circuit architecture which is inspired by the functional level evolution and Cartesian Genetic Programming (CGP). The whole intrinsic evolvable hardware system which includes fitness value calculation unit (which evaluates the quality of the filtered image), evolution algorithm unit (which generates the training bit-streams to optimize the architecture of the reconfigurable unit) and reconfigurable unit (which implement the evolution process by change its function and architecture) are implemented in the FPGA chip. The architecture of the whole evolution system is illuminated in Fig. 1.

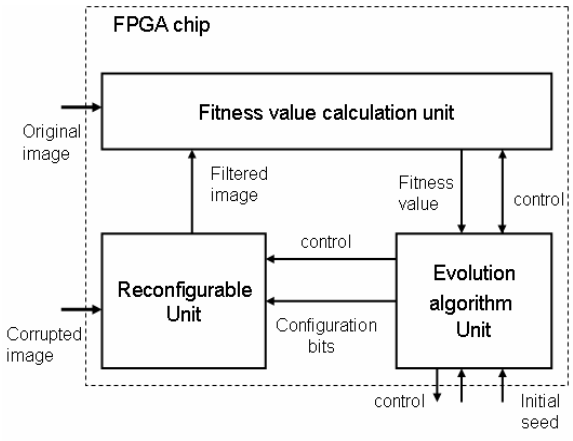


Fig.1 The framework of image filter design system

### 2.2 Reconfigurable circuit (phenotype)

According to Cartesian Genetic Programming, we employ an array of processing elements (PE) which is named as the reconfigurable circuit. The architecture of the reconfigurable circuit is shown in Fig. 2. The reconfigurable circuit includes nine 8-bit image input ports (labeled in Fig. 2 as I0-I8) and one 8-bit image output port. It means a pixel value of the image output is generated by using a $3 \times 3$ neighborhood mask ($3 \times 3$ input pixels from corrupted image which include the corresponding pixel of output pixel and its eight neighbors).It is shown in Fig. 3. By moving the $3 \times 3$ mask one pixel by one pixel in the pixels array of the corrupted image, the corrupted image is filtered. Totally, 49 PEs are placed in the Reconfigurable circuit and constitute an array of 8 rows and 7 columns.

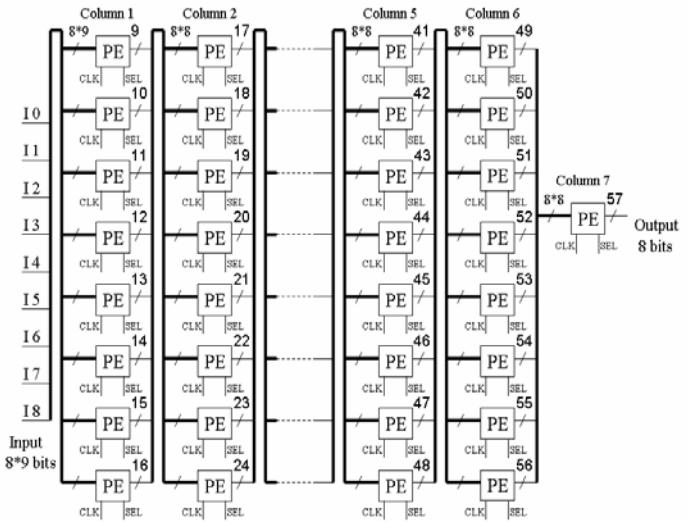


Fig.2 The reconfigurable circuit architecture of the EHW

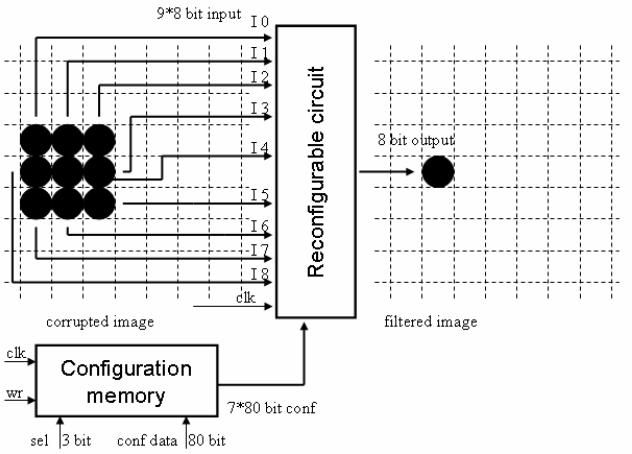


Fig. 3 Configuration memory and reconfigurable circuit

Each PE includes two 8 bit inputs and one 8 bit output. According to the Cartesian genetic programming, two inputs of PE can be connected to one of the outputs from the previous L columns (it is decided by L-back parameter which defines the level of connectivity and thus reduces/extends the search space.). In this paper, we define L=1 which means only neighboring column can be connected. In the array of PEs, each PE supports a certain functions which are applied to PE's two inputs (X and Y). These functions are listed in table 1, also shown in Fig. 4.

### 2.3 Configuration bit-streams (genotype)

The top level function of the reconfigurable circuit is defined by a chromosome which is a fixed size bit string. Each chromosome which is generated by the evolution algorithm unit contains a set of genes. Each gene corresponds to a PE in the reconfigurable circuit and is described by three values: the

position of two inputs of PE and the function implemented in PE.

Table 1. A list of functions implemented in processing element.

| Num | Function | Num | Function |
|---|---|---|---|
| 1 | X and Y | 8 | X |
| 2 | X or Y | 9 | Y |
| 3 | X xor Y | 10 | (X and 0F) xor (Y and F0) |
| 4 | X+Y | 11 | (X+Y+1)>>1 |
| 5 | (X+Y)>>1 | 12 | X>>1 |
| 6 | Max(X,Y) | 13 | X>>2 |
| 7 | Min(X,Y) | 14 | (not X) or Y |

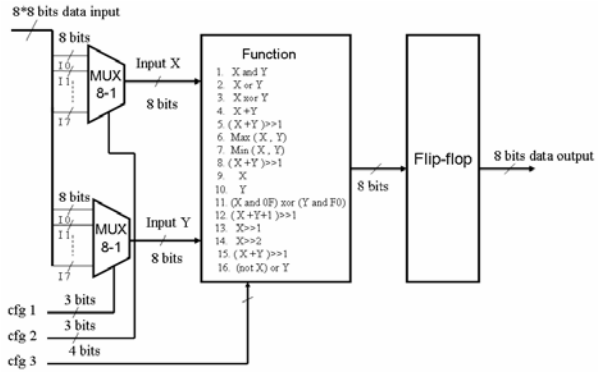


Fig. 4 The components of processing element

### 2.4 Evolution algorithm

The evolution operators in our experiment only include selection and mutation, crossover is not used. The initial population of four individuals is generated randomly according to the rules 90 and 150 as described by Wolfram [14]. After calculating the fitness values of the four individuals, the individual who has the highest fitness value is chosen as the parent of the next generation. Its four mutated versions provide the new population for the next generation. If one of the new individual has the better fitness than its parent, it substitutes its parent and be chosen as the new parent to generate the next generation. The evolutionary design process stops after 16384 ($2^{14}$) population generations.

### 2.5 Fitness function

To measure the quality of the filtered images, the MDPP (mean difference per pixel) fitness function is employed. The source image size was $k \times k$ pixels ($k = 256$), but only a sub area of $254 \times 254$ pixels could be considered because the pixel values at the image borders are ignored and thus remain unfiltered. The MDPP fitness function is described as fellow:

$$fitness_{MDPP} = 255 \times (k-2)^2 - DIFF \quad (1)$$

$$DIFF = \sum_{i=1}^{k-2}\sum_{j=1}^{k-2} |v(i,j) - w(i,j)| \quad (2)$$

In the Eq.(2), $v$ denotes the filtered image and $w$ denotes an original image which is uncorrupted by the noise.

## 3. HARDWARE IMPLEMENT OF THE IMAGE FILTER DESIGN AUTOMATION SYSTEM

### 3.1 Introduce of the experiment platform

The whole image filter design automation system is designed by using VHDL code and is implemented by using a Xilinx Virtex 2000E FPGA chip in a BG560 package which is fitting in the Celoxica RC1000 PCI bus plug-in board [15]. The RC1000 board is shown in Fig. 5. This board contains a PCI bridge that communicates between the RC1000 board and the host computer PCI bus. PCI bridge chip on the RC1000 board can operate correctly with PCI bus clocks maximum to 33MHz. Four banks of 2 Mbytes Static Random Access Memories (SRAM) for data processing operation are equipped. The Celoxica RC1000 board supports four 32-bit buses for the data communication between FPGA chip and SRAM.

Fig. 5 The RC1000 board with a Virtex 2000E FPGA chip

### 3.2 The reconfigurable unit

As the Fig. 3 shows, the reconfigurable unit is composed of the configuration memory and the reconfigurable circuit. The reconfigurable circuit operates nine 8bit image pixels input and one 8bit pixel output. By shifting the mask in the image pixels array, $254 \times 254$ pixels in the corrupted image can be processed by the reconfigurable circuit.

The reconfigurable circuit employs 49 PEs constituting the PEs array of 8 rows and 7 columns. The last column only includes one PE. Pipeline technique is employed in our experiment. A column of PEs is considered as a stage of the pipeline process. Each PE in the reconfigurable circuit is made up of two 8-bit input multiplexers, one Functional Block (FB) and one flip-flop as shown in Fig. 4. Flip-flop is equipped for supporting the pipeline processing. The FB contains 14 different functions which are listed in table 1. One of the functions is to be chosen as the active function of PE at one system clock cycle. For each PE, the multiplexer inputs are connected to the outputs of PEs of the previous column. In column 2, 3, 4, 5, 6, 7, it is obvious that each 8-bit input multiplexers of the PE can connect to the eight outputs of previous column's PEs. In column 1, unfortunately the reconfigurable circuit has nine 8-bit inputs data path. So the first 8-bit input multiplexer of PE in column 1 is constrained to be connected with circuit's inputs I0-I7 and the second 8-bit input multiplexer is constrained to be connected with circuit's inputs I1-I8. Each PE needs 3+3+4 address bits to decide its input connection and active function.

The configuration memory is employed to store the configuration bit-streams. In the pipeline process, Eight PEs in the same column need to be configured simultaneously, so the configuration memory is divided into seven blocks and each block's 80 bits configuration bit-streams can configure a column of PEs. Although the number of configuration bits for the last column of PE is 10 bits, the configuration memory still employs a 80 bits block in order to simply the circuit design. Totally, the configuration memory consists of $7 \times 80$=560 bits.

### 3.3 Fitness value calculation unit

The architecture of fitness value calculation unit is detailed in Fig. 6. In the hardware implement phase, instead of implementing the MDPP function Eq. (1) in the hardware, we only calculate the DIFF value Eq. (2). The fitness value calculation unit includes an absolute value subtracter which evaluates the difference of the correlated pixels between the filtered image and original image; an accumulator which accumulates the output of the absolute value subtracter. Getting the DIFF value of the image needs to calculate $254 \times 254$ pixels in the filtered image.

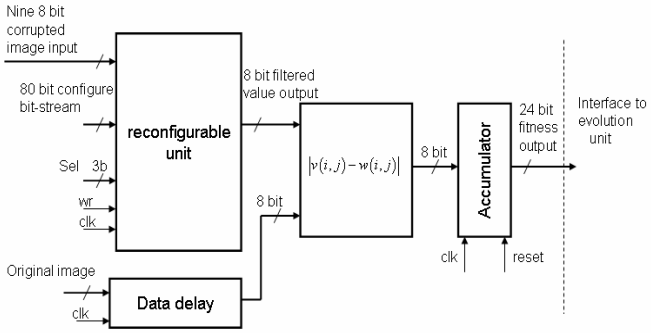


Fig. 6. Fitness value calculation

### 3.4 Evolution algorithm unit

The evolution algorithm unit is also implemented in the same FPGA chip for the purpose of speedup the evolution process. The evolution algorithm unit mainly includes the best chromosome memory (7×80 bits), the best fitness value memory (24 bits), system control unit, population memory (which consists four 7×80 bits individual memory), random number generator and mutation unit as shown in Fig. 7.

**Random number generator (RNG):** the random number generator consists of 80 alternating cells which change their state according to rules 90 and 150 as described by Wolfram [14]. After loading the initial seed, the RNG can generate a sequence of random bit strings. The RNG generates two kinds of random bit strings as the output. The 80 bit RNG output is treated as the prototype of the four individuals of the initial population. The 7 bit RNG output is supplied to mutation module to decide the mutation points of each chromosome.

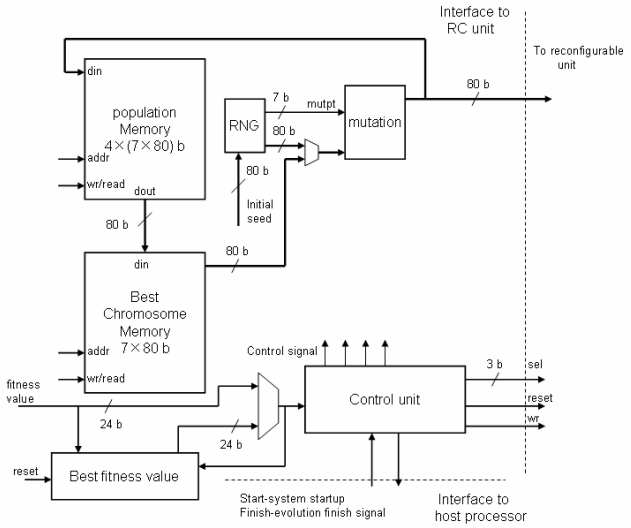


Fig. 7. The architecture of evolution algorithm unit

**Mutation:** for each chromosome, mutation unit read the mutation position parameter from the RNG and the mutation data input coming from the best chromosome memory or RNG. For each integrated individual, in average 5 bits are mutated.

**System control unit:** The system control unit communicates between outside environment and image filter design automation system. Another function of the system control unit is to control the operations of the modules in the system.

## 4. EVOLUTION PROCESS OF THE IMAGE FILTER DESIGN AUTOMATION

The image filter evolution process begins after the design automation system getting the start signal from the host processor. An 80-bit random initial seed is sent to RNG from the host processor. The four individuals of the first generation are generated by the mutation module which operates the output of the RNG. Each individual ($7 \times 80$ bit) which is generated by the evolution algorithm unit is treated as the configuration bit-streams of the reconfigurable circuit. Because of the pipeline process, it takes 7 clock cycles to configure the array of PEs which includes seven columns. Simultaneously, the inputs of the reconfigurable (nine 8-bit pixels of the corrupted image) are processed by the array of PEs as a pipeline process. In our experiment, the parameter L=1, it takes 7 clock cycles to get the first pixel output of the filtered image. Because of the pipeline process, after the first pixel output of the filtered image, we get one pixel output in each clock cycle. The intact corrupted image is processed in $254 \times 254+6$ clock cycles.

After the fitness value of the four individuals of the first generation is evaluated, the chromosome which has the best fitness value can be chosen as the parent of the next generation and stored in the best chromosome memory. The four mutated versions of the best chromosome are treated as the individuals of the next generation. Because the time consumed in configuring the reconfigure circuit is hided in the in the image filtering process, the full evolution time is determined as:

$$t = \frac{(k-2)^2 \times \mu \times n}{f} \tag{3}$$

In the Eq. (3), $(k-2)^2$ is the number of processed pixels, $\mu$ is the number of individuals in each generation, $n$ is the number of generations, $f$ is the operation frequency of the hardware platform.

## 5. EXPERIMENT RESULT

### 5.1 Synthesis report

Before generating the final FPGA configuration file, our design is synthesized into Xilinx Virtex 2000E FPGA using Xilinx Integrated Software Environment. The synthesis result is optimized for speed. The reconfigurable circuit occupies 28% resource of the Xilinx Virtex 2000E (5547 slices). The whole image filter design automation system uses 45% resource of the Xilinx Virtex 2000E (8642 slices). According to the synthesis report, the maximal operational frequency of the system is 69.013MHz.

### 5.2 Hardware evolution result

The goal of our experiment is to evolve a digital image filter for a kind of additive image noise. Only gray-scale (8-bit each pixel) image of size $256 \times 256$ pixels is considered. In

our experiment, two types of additive noise which is independent of the image itself are processed: Gaussian noise and shot noise (also called "salt & pepper" noise). Both Gaussian noise and shot noise are generated by using Matlab function: imnoise (A , 'gaussian', 0, 0.008) (generating Gaussian noise of mean 0 and variance 0.008) or imnoise (A , 'salt & pepper', 0.05) (denoting the image contaminated by shot noise with 5% of corrupted pixels). Two types of image filters are evolved to process the above additive image noises independently. The evolution process is individually trained by using Lena (a popular image for testing) and Lena with Gaussian noise of mean 0 and variance 0.008 or using Lena and corrupted Lena in which 5% pixels are destroyed by the shot noise.

Only the sub area of $254 \times 254$ pixels in the corrupted image is filtered because the pixels at the border can not be processed by the $3 \times 3$ mask. The evolved filter is the evolutionary result of the array of PEs after 16384 generations. In the actual hardware evolution process of our experiment, the image filter design automation system which is based on the Xilinx Virtex 2000E FPGA chip can operate steadily under 9 MHz. Hardware took less than 8 minutes to get an evolved image filter result. Some image filters which were evolved to processed Gaussian and shot noise are shown in table 2. As a comparing, a set of conventional image filter are also listed in table 2. Two evolved circuit results ES1 and EG1 are shown in Figs. 8.1~8.2. In the Figs. 8.1~8.2, the blocks such as F7, F6, F5 are the functions which are listed table 1.

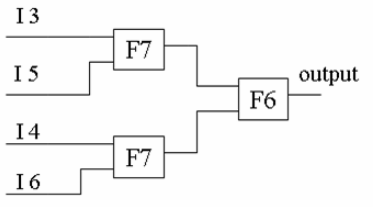


Fig. 8.1. The evolved circuit for shot noise (ES1)

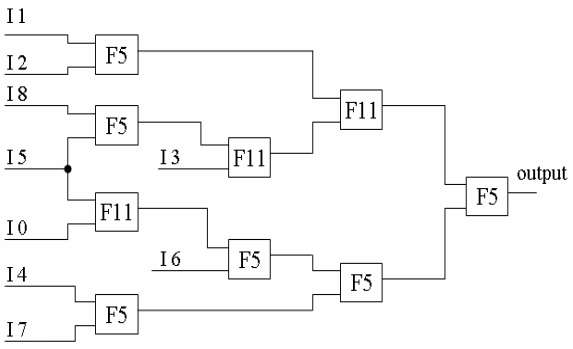


Fig. 8.2. The evolved circuit for gaussian noise (EG1)

Faced the additive shot noise, Figs. 9.1~9.6 present a series of images to contrast the quality of the filtered images which are processed by different image filters. Fig. 9.1 is the original Lena image, which in Fig. 9.2 is destroyed by 5% shot noise. Fig. 9.3~9.5 are processed results by median filter, FA4 filter and mean filter. Fig. 9.6 is the result filtered by the ES1 EHW filter.

Some samples of filtered results of the Lena image including Gaussian noise are also exhibited. Fig. 10.1 is the original Lena image. Corrupted Lena by Gaussian noise of mean 0 and variance 0.008 is Fig. 10.2. Fig 10.3~10.6 are the filtered results from median filter, FA4 filter, mean filter and EG1 EHW filter.

Table 2. Evolved image filter and conventional image filter (median, FA4, mean). Symbols used: TFN trained for noise, G8 Lena with Gaussian noise of mean 0 and variance 0.008, S5 Lena of 5% pixels are corrupted by the shot noise.

| Filter | TFN | DIFF |
|---|---|---|
| median | G8 | 602767 |
| FA4 | | 669243 |
| mean | | 544941 |
| EG1 | G8 | **541003** |
| EG2 | | 682332 |
| EG3 | | 683133 |
| EG4 | | 676363 |
| median | S5 | 226859 |
| FA4 | | 744435 |
| mean | | 601803 |
| ES1 | S5 | 355622 |
| ES2 | | 256868 |
| ES3 | | 243772 |
| ES4 | | **223576** |

Fig. 9.1 Original Fig. 9.2 Corrupted Fig. 9.3 Median

Fig. 9.4 FA4 Fig. 9.5 Mean Fig. 9.6 EHW

We also employed various image sets baboon and cameraman to test the generality of the evolved image filter which is trained by Lena and Lena with different noises. The result is listed in table 3.

Fig. 10.1 Original Fig. 10.2 Corrupted Fig. 10.3 Median


Fig. 10.4 FA4


Fig. 10.5 Mean


Fig. 10.6 EHW

Table 3. The generality of evolved image filter. Columns 3-5 present DIFF for a given filter and image. Average reports the average DIFF of evolved image filter for all test images.

| filter | N | Lena | baboon | camera man | average |
|---|---|---|---|---|---|
| median | G8 | 602767 | 1168182 | 651480 | 807476 |
| FA4 | | 669243 | **1079493** | 701174 | 816637 |
| Mean | | 544941 | 1134862 | 634649 | 771484 |
| EG1 | | **541003** | 1119510 | **627520** | **762677** |
| median | S5 | **226859** | 962943 | **285936** | **491913** |
| FA4 | | 744435 | 1124378 | 817748 | 895520 |
| Mean | | 601803 | 1180275 | 677402 | 819826 |
| ES1 | | 355622 | **864969** | 360734 | 527108 |

### 5.3 Discussion

Our hardware platform can run steadily under 9 MHz and use less than 8 minutes to get an evolved result. According to Yang Zhang's report [13], software simulations (PentiumII 266 CPU based) take more 15 hours and his hardware evolution process spends 95 minutes to achieve the result. Considering the different of the evolution generations and individuals between us, our evolution speed is still improved one order of magnitude over Yang Zhang's intrinsic evolution which executes the genetic operation in host processor.

All of the results in table 2 are based Lena. We can find that some evolved filters exhibit lower DIFF value than conventional image filter and the evolved EHW filters ES1 and EG1 express a good generality in table 3.

## 6. CONCLUSION

In this paper, we approach the problem of the image filter design automation by using a kind of intrinsic evolution method which employs the reconfigurable circuit architecture as the reconfigurable component of the EHW. The basic concept and hardware implement method of the evolutionary image filter design system are clarified. Especially, the architecture of single common FPGA chip implement of the whole evolution design system is applied in our experiment, which is due to speedup the hardware evolution process and bring the feasibility of embedded system solution. Celoxica Rc1000 Board based hardware experiment result proves that our intrinsic evolution method can successfully evolve different efficient image filters faced with different additive image noises and our complete hardware evolution architecture can efficiently speedup the evolution process. The future work will be devoted to employ this kind of intrinsic evolution architecture in other EHW application fields such as adaptation EHW domain.

## REFERENCES


[1] Yao, X ., Higuchi , T .: Promises and Challenges of Evolvable Hardware. In : Proc. Of the 1st International Conference on Evolvable Systems: From Biology to Hardware ICES'96, LNCS 1259, Springer-Verlag , Berlin (1997) 55-78.

[2] Murakawa , M . et al.: Hardware Evolution at Function Level. In : Proc. Of the Parallel Problem Solving from Nature PPSN IV , LNCS 1141, Springer-Verlag , Berlin (1996) 62-72.

[3] T. Higuchi, M. Iwata, I. Kajitani, H. Iba, T. Furuya, and B. Manderick, "Evolvable hardware and its applications to pattern recognition and fault-tolerant systems," in Toward Evolvable Hardware: The Evolutionary Engineering Approach, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 118–135.

[4] H. de Garis.,"LSL evolvable hardware workshop report" ATR, Japan,Tech. Rep., Oct. 1995.

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

[6] Hollingworth, G., Smith, S., Tyrrell, A.: The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic. In: Proc. Of the 3rd International Conference on Evolvable Systems: From Biology to Hardware ICES'00, LNCS 1801, Springer-Verlag, Berlin (2000)72-79.

[7] Hollingworth, G.; Smith, S.; Tyrrell, A.: Safe intrinsic evolution of Virtex devices. In The Second NASA/DoD Workshop on Evolvable Hardware, 13-15 July 2000, Pages:195 – 202.

[8] Xilinx Inc., XC6200 Field Programmable Gate Arrays Datasheet V1.10, San Jose, CA, 1997.

[9] S. A. Guccione, D. Levi, and P. Sundararajan. JBits: A java-based interface for reconfigurable computing. In 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD), 1999.

[10] Julian F. Miller, Peter Thomson.: Cartesian Genetic Programming In: Proc. Of the Third European Conference on Genetic Programming. LNCS, Vol. 1802, (2000) pp.121-132, Springer-Verlag.

[11] L. Sekanina, V. Drabek: Automatic Design of Image Operators Using Evolvable Hardware. Fifth IEEE Design and Diagnostic of Electronic Circuits and Systems DDECS'02, 132- 139.

[12] L. Sekanina: Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. Evolvable Systems: From Biology to Hardware. Fifth International Conference, ICES 2003, 186-198.

[13] Y. Zhang , S. Smith and A.Tyrrell .: Digital Circuit Design Using Intrinsic Evolvable Hardware. In: Proc. Of the 2004 NASA/DoD Conference on the evolvable Hardware, IEEE Computer Society, Los Alamitos (2004) 55-63.

[14] S. Wolfram. Universality and complexity in cellular automata. Physica, 10D:1-35, 1984.

[15] Celoxica Inc., RC1000 Hardware Reference Manual V2.3, 2001.