Genetic Algorithm Processor for Image Noise Filtering Using Evolvable Hardware

Dr. K. Sri Rama Krishna srk_kalva@yahoo.com
Professor & HOD /ECE
V R Siddartha Engg College
Vijayawada, Krishna (dist), A.P-520007, India

A. Guruva Reddy guruvareddy78@mail.com.com
Professor / ECE
NRI institute of Technology
Pothavarappadu , Agiripalli, Krishna (D), A.P-521 212, INDIA

Dr. M.N.GIRI PRASAD mahendran_gp@rediffmail.com
Principal,
JNTU college of Engineering,
Pulivendala, Kadapa (dist). A.P – 516390, India

Dr. K.Chandrabushan Rao cbraokota@yahoo.com
Professor/ECE
MVGR College of Engineering,
Vijayanagarm, A.P, India

M. Madhavi madhavi.418@gmail.com
Associatet Prof / ECE
NRI institute of Technology
Pothavarappadu , Agiripalli, Krishna (D), A.P-521 212, INDIA

Abstract

General-purpose image filters lack the flexibility and adaptability of un-modeled noise types. On the contrary, evolutionary algorithm based filter architectures seem to be very promising due to their capability of providing solutions to hard design problems. Through this novel approach, it is made possible to have an image filter that can employ a completely different design style that is performed by an evolutionary algorithm. In this context, an evolutionary algorithm based filter is designed in this paper with the kernel or the whole circuit for automatically evolved.

The Evolvable Hard Ware architecture proposed in this paper can evolve filters without a priori information. The proposed filter architecture considers spatial domain approach and uses the overlapping window to filter the signal. The approach that is chosen in this work is based on functional level evolution whose architecture includes nonlinear functions and uses genetic algorithm for finding the best filter configuration.

Keywords: Reconfigurable hardware, Processing elements, Genetic algorithm, Virtual Reconfigurable Circuit

1. INTRODUCTION

Non-linear Image processing [13] with good flexibility and adaptability is highly desirable in several applications such as image transformation, correction of distortion effects, noise removal, histogram equalization etc. Conventional adaptive filter lacks the flexibility for adapting to changes in architecture and is therefore suitable for compensating non-uniform variations in Signal-to-Noise Ratio (SNR). It is also reported that conventional adaptive filter performs well, only when, the spatial density of the noise is less. In this paper, a reconfigurable computing FPGA architecture is designed for adapting to the changes in task requirements or changes in environment, through its ability to reconfigure its own hardware structure dynamically and autonomously with design objectives such as high performance, specialization and adaptability.

In this paper, minimally sufficient hardware resources are dynamically allocated based on noise levels at specific time intervals. To enable automatic recovery of a device after damage, an autonomous restructuring algorithm to handle internal processing element fault is implemented and tested successfully. The novelty of this work is the implementation of a reconfigurable system [1] and an associated design methodology that has both flexibility and autonomous restructuring [3] of Processing Elements. The reconfiguration process is achieved using an evolutionary algorithm [2] such as Genetic Algorithm (GA).

2. IMAGE ENHANCEMENT USING EVOLUTIONARY DESIGN

The Evolvable Hard Ware (EHW) architecture [6] proposed in this work for filtering the noise present in the image that is subsequently realized on an Field Programmable Gate Array (FPGA) based image processing board, consists of the GA processor [11] and a virtual reconfigurable circuit and is shown in Figure 1. This type of implementation integrates a hardware realization of genetic algorithm and a reconfigurable device. These two modules of the EHW [5] are described in the following sections:

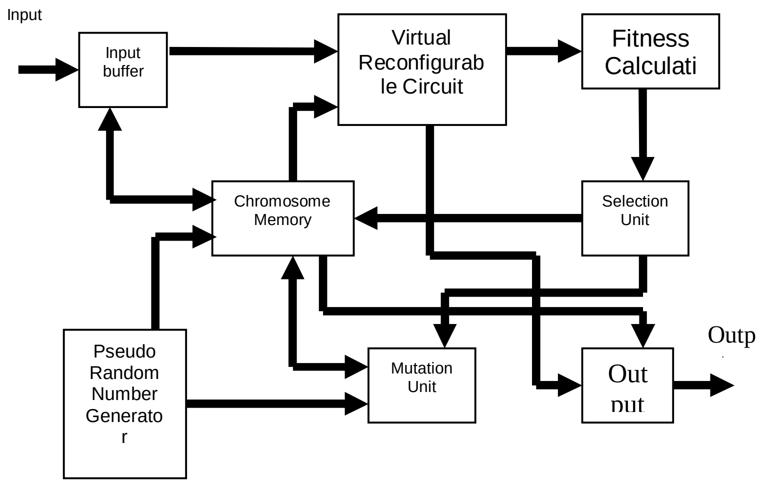


FIGURE 1: EHW chip with the VRC and GA Processor

2.1 Implementation of the GA Processor

The implementation of simple GA [12] is composed of following basic modules: pseudo random number generator, population memory, selection unit, mutation unit, fitness evaluator and output buffer. These modules have already been discussed in chapter II of this thesis.

2.2 Implementing the Virtual Reconfigurable Circuit

The Virtual Reconfigurable Circuit (VRC) is implemented [10] as a combinational circuit using the concepts of pipelining. It consists of processing elements arranged in rows and columns. In this work, a total of 25 PE's are selected and are arranged in six rows and four columns with the 25th PE representing the final output.

2.3 Architecture of The VRC

The architecture of the VRC is shown in Figure 2. I_4 represents the filtered output from the VRC. The nature of operation performed on the input depends on the configuration bits downloaded into the configurable memory from the genetic unit.

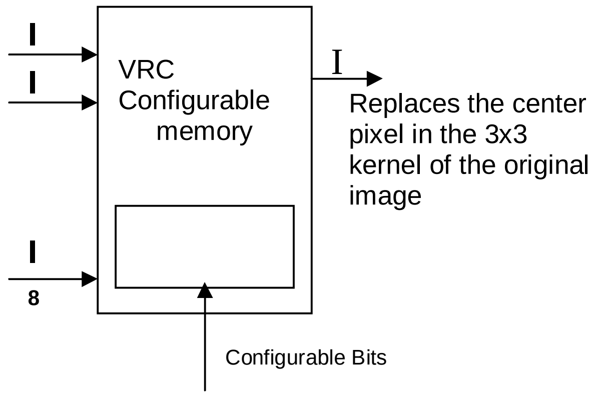

I
I
I
8
VRC
Configurable
memory
I
Replaces the center pixel in the 3x3 kernel of the original image
Configurable Bits

FIGURE 2: Architecture of the VRC

3. GENETIC ALGORITHM FOR EVOLVABLE HARDWARE

The configuration bits are obtained using genetic algorithm and are downloaded into the reconfiguration circuit which results in relocation of hardware modules inside VRC. The flow diagram of reconfiguration process is shown in figure 3.

The class of evolutionary algorithm most commonly used in evolvable hardware [4] is the genetic algorithm. Most commonly these operate on a fixed size population of fixed length binary strings called chromosomes. Each chromosome encodes a common set of parameters that describe a collection of electronic components and their interconnections, thus each set of parameter values represents an electronic circuit. The set of all possible combinations of parameter values defines the search space of the algorithm, and the circuits that they represent define the solution space of the algorithm.

The Evolvable hardware with Genetic algorithm [7] begins by initialising the bits of each chromosome with random values. The chromosomes are then evaluated in turn by creating a circuit based on the parameter values, either as a simulated model of the circuit or as a concrete circuit embodied in reconfigurable hardware . The circuit's fitness for performing the target task is then measured by passing it a set of test values and evaluating the veracity of the circuit's output. The selection operator then probabilistically populates the next generation of chromosomes such that chromosomes with high fitness are more likely to be selected. The operator selects two individuals at random and compares their fitness. Only the individual with the highest fitness is inserted into the next generation. If they have equal fitness the individual to be inserted is chosen at random. Once the new population has been selected, it is varied. Common variation operators are one-point crossover and point mutation. One point crossover recombines two chromosomes by choosing a position at random along the chromosome and swapping every bit beyond this point between the strings. It is stochastically applied according to a fixed probability. Point mutation independently inverts each bit in the chromosome according to a fixed probability. These operators are applied to all members of the new population. Often in addition to these operators, the best member of the original population is copied into the new population unchanged, The new population is now complete and the algorithm then iterates the steps of evaluation, selection and variation until a circuit that functions adequately is found, or a pre-specified number of generations is completed.

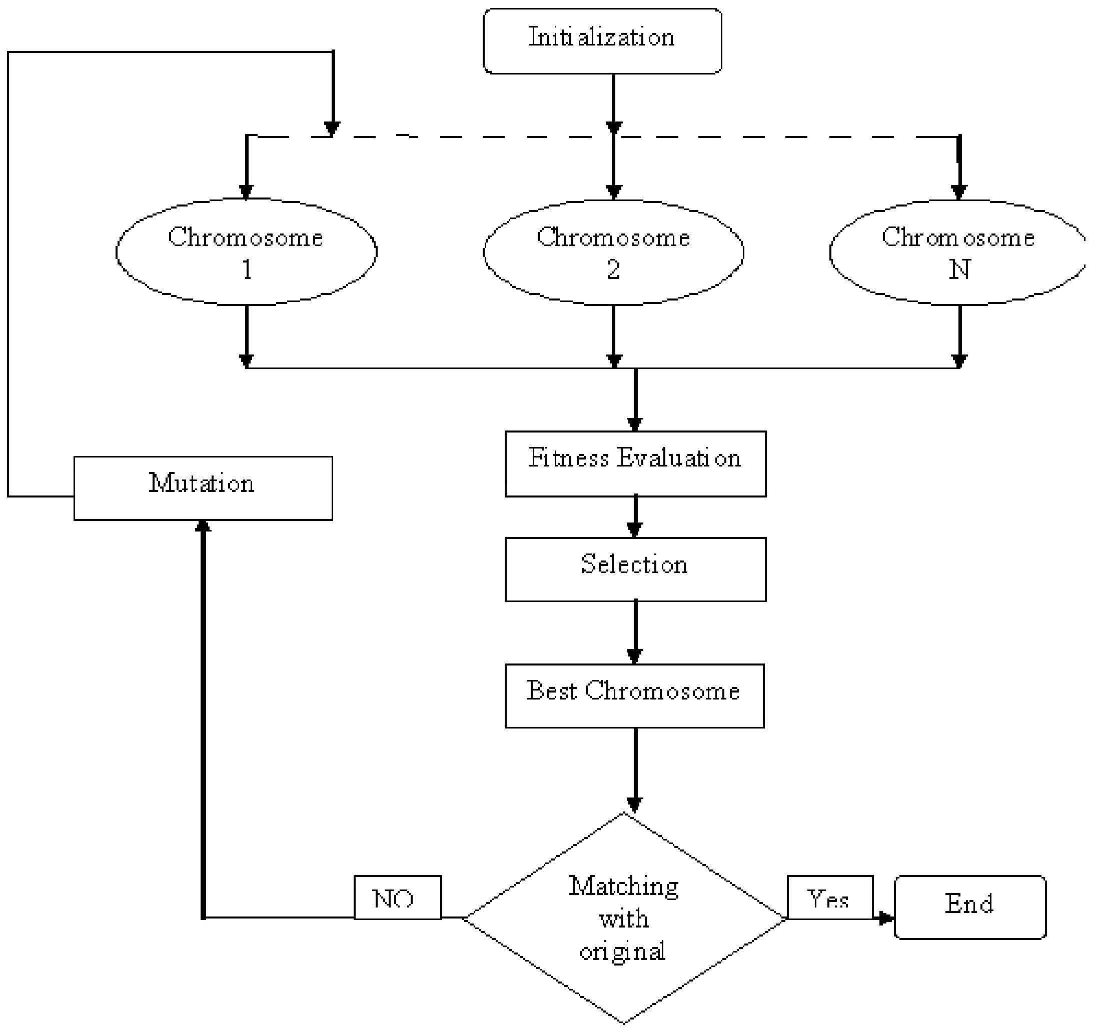


FIGURE 3: Flow diagram of VRC using Genetic Algorithm.

The configuration bits are obtained using genetic algorithm [12] and are downloaded into the reconfiguration circuit which results in relocation of hardware modules inside VRC.

4. COMPONENTS OF VIRTUAL RECONFIGURABLE CIRCUIT

In reconfigurable computing sequence of configurations is not known at the design time. It involves the usage of software and hardware components normally referred to as IP cores. This concept is very much useful in effective designing of the complex systems. The reconfigurable device is part of the evolvable component. The genetic unit in the evolvable component generates the configuration bits (Chromosomes). The fitness calculation is usually performed outside the evolvable component. Internal reconfiguration implies that a CLB can configure other CLB's.

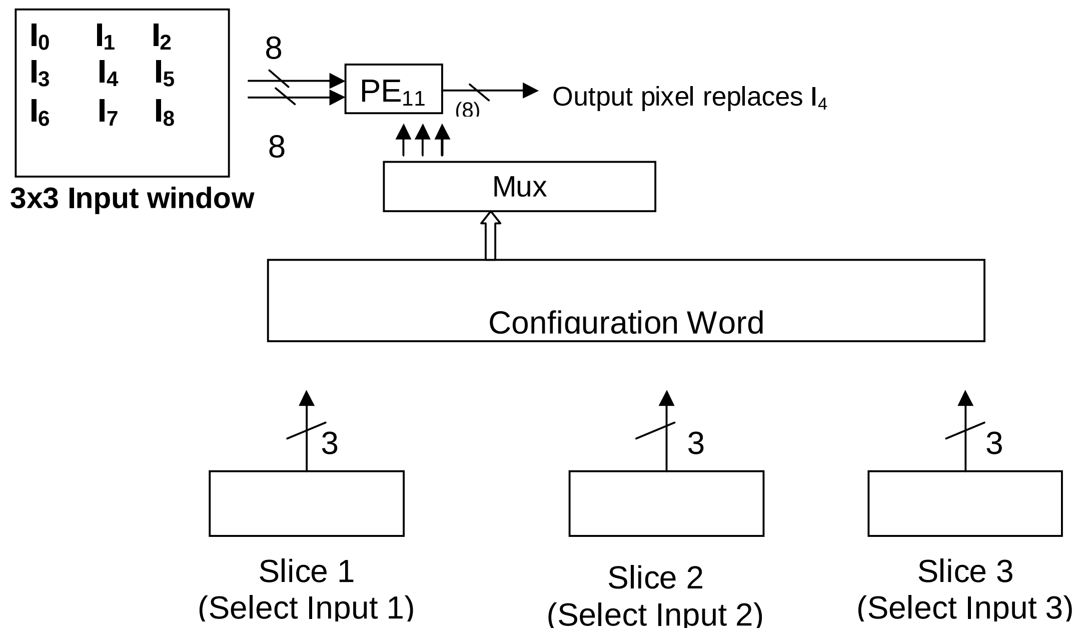


FIGURE 4: VRC with the internal MUX for selecting inputs and functions Helvetica

In figure 4. Slice 1 is a ‘m’ bit vector and selects any one of many inputs and assigns it as first input X

Slice 2 is a ‘m’ bit vector and selects any one of many inputs and assigns it as second input Y

Slice 3 is a ‘n’ bit vector and selects any one of the 16 functions to be performed on X and Y

The X and Y are both 8-bit vectors and the processed output is also an 8 - bit vector.

5 IMAGE ENHANCEMENT ALGORITHM

5.1 Fitness Function

Popular measures of performance for evaluating the difference between the original and filtered images includes

i. Peak Signal to Noise Ratio (PSNR) and
ii. Mean Difference per Pixel (MDPP)

In many applications the error is expressed in terms of a signal-to-noise ratio (SNR), and is given in equation 1

$$\mathrm{SNR} = 10\log_{10}\frac{\sigma^2}{\mathrm{MSE}}\ \ \mathrm{dB} \tag{1}$$

where σ^2 is the variance of the desired or original image. The peak signal-to-noise ratio (PSNR) is expressed as equation 2,

$$\mathrm{PSNR} = 10\log_{10}\frac{255^2}{\mathrm{MSE}}\ \ \mathrm{dB} \tag{2}$$

The fitness function using MDPP is given by

$$\mathrm{MDPP} = \frac{1}{\mathrm{NxN}}\sum_{i,\,j=1}^{N}|\,\mathrm{orig}(i,\,j) - \mathrm{filt}(i,\,j)\,| \tag{3}$$

where |orig(i,j) – filt(i,j)| is the absolute difference between the original and filtered images In this work, the Mean Difference per Pixel (MDPP) is used as a performance measure by the fitness evaluator module in the GA processor for the reason that MDPP fitness function is computationally easier for hardware implementation in comparison to PSNR. The EHW architecture that has the best MDPP (minimum MDPP) after a specified number of generations is chosen as the evolved architecture.

6 NOISE MODELS

6.1 Multiplicative Noise

In this model, the noise magnitude depends on the signal magnitude itself. An example of multiplicative noise is the degradation of film material caused by the finite size of silver grains used in photosensitive emulsion.

6.2 Quantization Noise

This occurs when insufficient quantization levels are used, for example, only 50 levels for a monochromatic image. Due to this noise false contours appear. However, this noise can be eliminated by simple means.

6.3 Impulsive Noise

When an image is corrupted with individual noisy pixels whose brightness differs significantly from that of the neighborhood, then it represents an impulse noise effect.

6.4 Salt and Pepper Noise

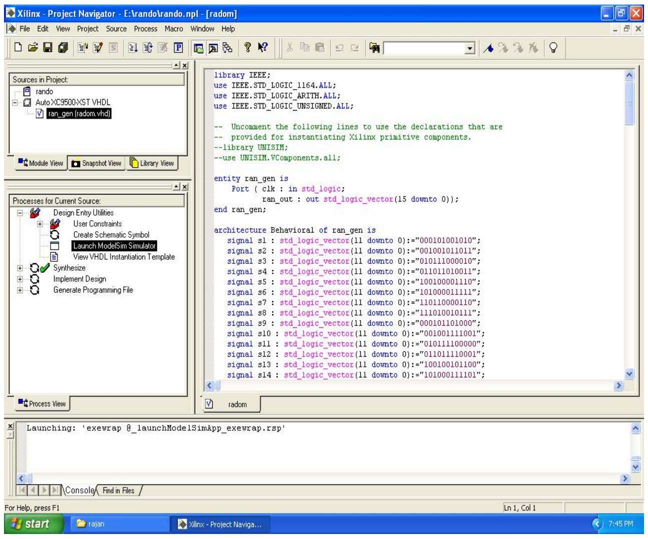
This describes [8] saturated impulsive noise – an image corrupted with white and/or black pixels. The effect is more appropriate for binary images.

6.5 Gaussian Noise

Idealized noise, called white noise or Gaussian noise, has constant power spectrum, meaning that its intensity does not vary with increasing frequency. This noise model is often used.

7. VHDL IMPLEMENTATION OF GA PROCESSOR

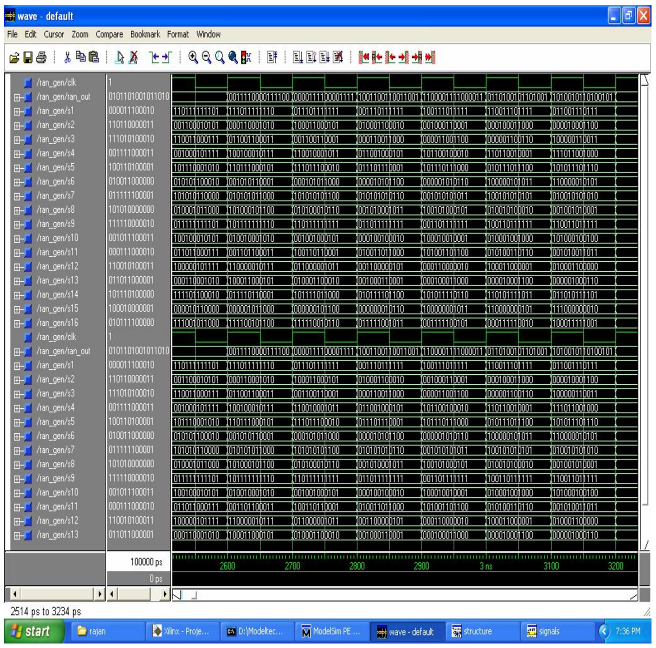
The model of GA processor is implemented in VHDL. The screen that captured the VHDL code from the Xilinx ISE software package [9] is given in Figure 5. Each one of the signal implements, the 12 bit random number generator and a sample output captured using the modelsim package, is shown in Figure 6. These bits (GA processor output) are the configuration bits which control the interconnections among the PE's as well as and also the functionality of each PE in the VRC module [10].



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity ran_gen is
    Port ( clk : in std_logic;
           ran_out : out std_logic_vector(15 downto 0));
end ran_gen;

architecture Behavioral of ran_gen is
   signal s1 : std_logic_vector(11 downto 0):="000101001010";
   signal s2 : std_logic_vector(11 downto 0):="001001011011";
   signal s3 : std_logic_vector(11 downto 0):="010111000010";
   signal s4 : std_logic_vector(11 downto 0):="011011010011";
   signal s5 : std_logic_vector(11 downto 0):="100100001110";
   signal s6 : std_logic_vector(11 downto 0):="101000011111";
   signal s7 : std_logic_vector(11 downto 0):="110110000110";
   signal s8 : std_logic_vector(11 downto 0):="111010010111";
   signal s9 : std_logic_vector(11 downto 0):="000101101000";
   signal s10 : std_logic_vector(11 downto 0):="001001111001";
   signal s11 : std_logic_vector(11 downto 0):="010111100000";
   signal s12 : std_logic_vector(11 downto 0):="011011110001";
   signal s13 : std_logic_vector(11 downto 0):="100100101100";
   signal s14 : std_logic_vector(11 downto 0):="101000111101";
```

```
Launching: 'exewrap @_launchModelSimApp_exewrap.rsp'
```

FIGURE 5: Implementation of GA processor using VHDL



```
/ran_gen/clk        1
/ran_gen/ran_out    0101101001011010
/ran_gen/s1         000011100010
/ran_gen/s2         110110000011
/ran_gen/s3         111010100010
/ran_gen/s4         001111000011
/ran_gen/s5         100110100001
/ran_gen/s6         010011000000
/ran_gen/s7         011111100001
/ran_gen/s8         101010000000
/ran_gen/s9         111110000010
/ran_gen/s10        001011100011
/ran_gen/s11        000111000010
/ran_gen/s12        110010100011
/ran_gen/s13        011011000001
/ran_gen/s14        101110100000
/ran_gen/s15        100010000001
/ran_gen/s16        010111100000
/ran_gen/clk        1
/ran_gen/ran_out    0101101001011010
/ran_gen/s1         000011100010
/ran_gen/s2         110110000011
/ran_gen/s3         111010100010
/ran_gen/s4         001111000011
/ran_gen/s5         100110100001
/ran_gen/s6         010011000000
/ran_gen/s7         011111100001
/ran_gen/s8         101010000000
/ran_gen/s9         111110000010
/ran_gen/s10        001011100011
/ran_gen/s11        000111000010
/ran_gen/s12        110010100011
/ran_gen/s13        011011000001

                    100000 ps
                    0 ps
2514 ps to 3234 ps
```

FIGURE 6: Modelsim Captured GA Processor output

8. IMPLEMENTATION RESULT OF EHW FILTER

In this section, the performance of EHW Filter in removing Gaussian noise is presented. The evolved EHW architecture for this case is shown in figure 7.

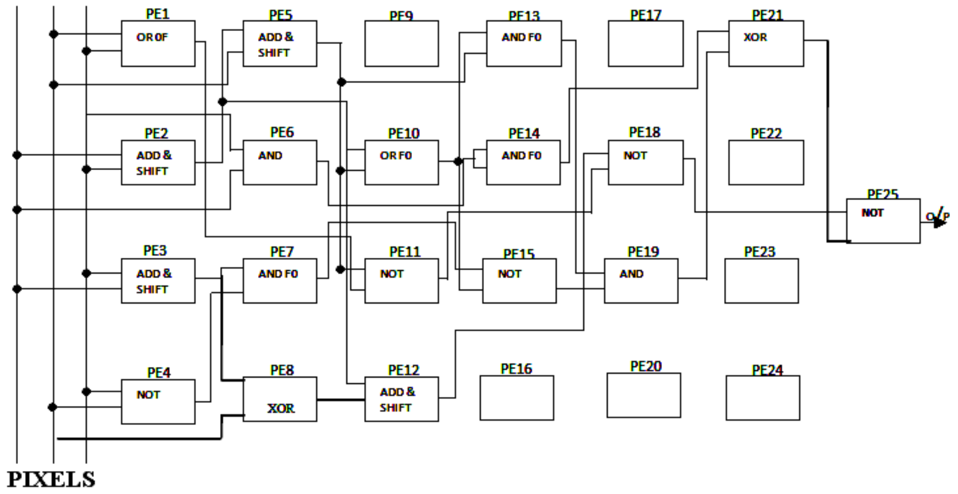


FIGURE 7: Architecture of VRC evolved to filter Gaussian Noise

The standard test image, Lena as shown in figure 8a is used for study and the results. In figure 8b shows Additive Gaussian noise with mean 0 and standard deviation 0.05 is added. The results obtained with Median filter and proposed EHW filter are shown in Figure 8c and 8d respectively. It is observed that Median Filter does not effectively remove Gaussian noise and performs only in edge regions, with blur effects still present in continuous regions. The EHW Filter performed well both in edge and continuous regions and effectively removed Gaussian noise.

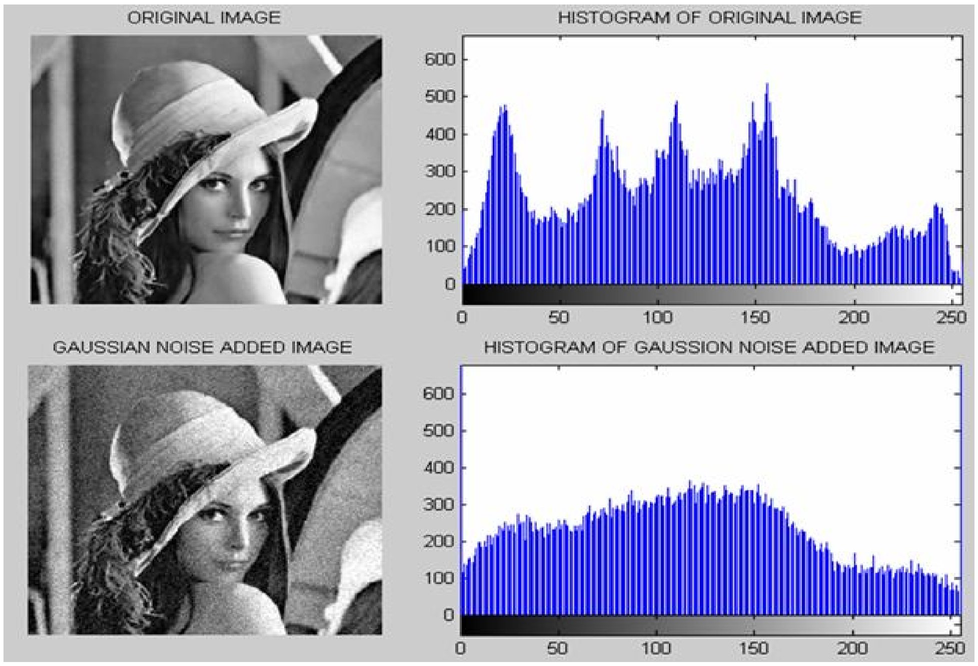


FIGURE 8 a: Original Image,
FIGURE 8 b: Gaussian noise added image

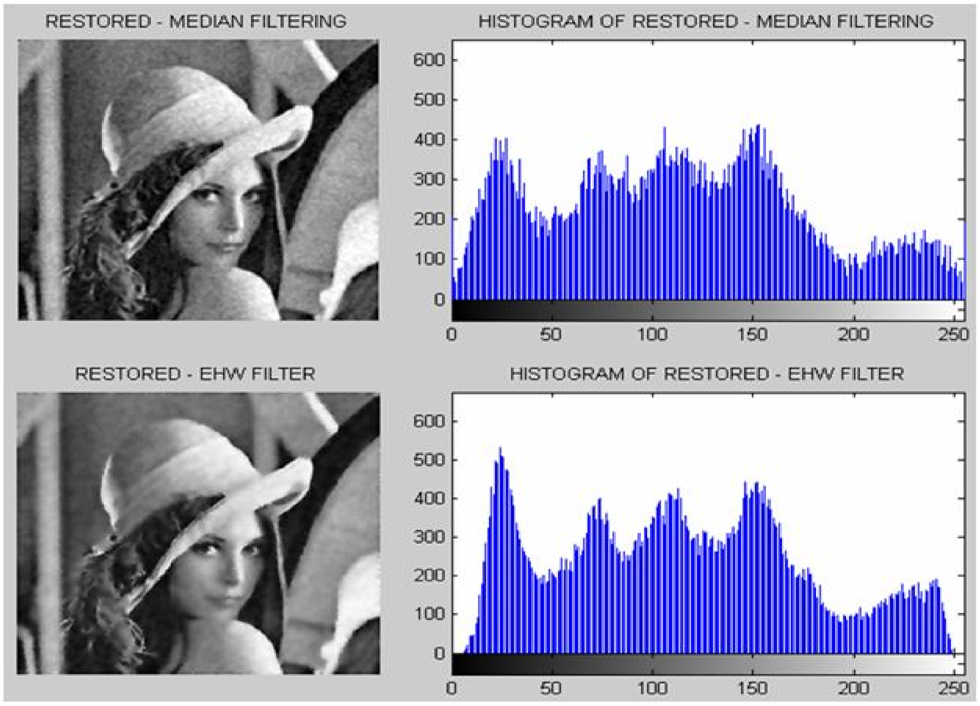


FIGURE 8 c: Restored image using Median
FIGURE 8 d: Restored image using EHW

The standard test image, Lena as shown in figure 9a. In figure 9b shows Salt And Pepper is added for Lena image. The results obtained with sober filter and proposed EHW filter are shown in Figure 9c and 9d respectively. It is observed that sober filter does not effectively remove salt and pepper noise. The EHW Filter performed well both in edge and continuous regions and effectively removed salt and pepper noise.

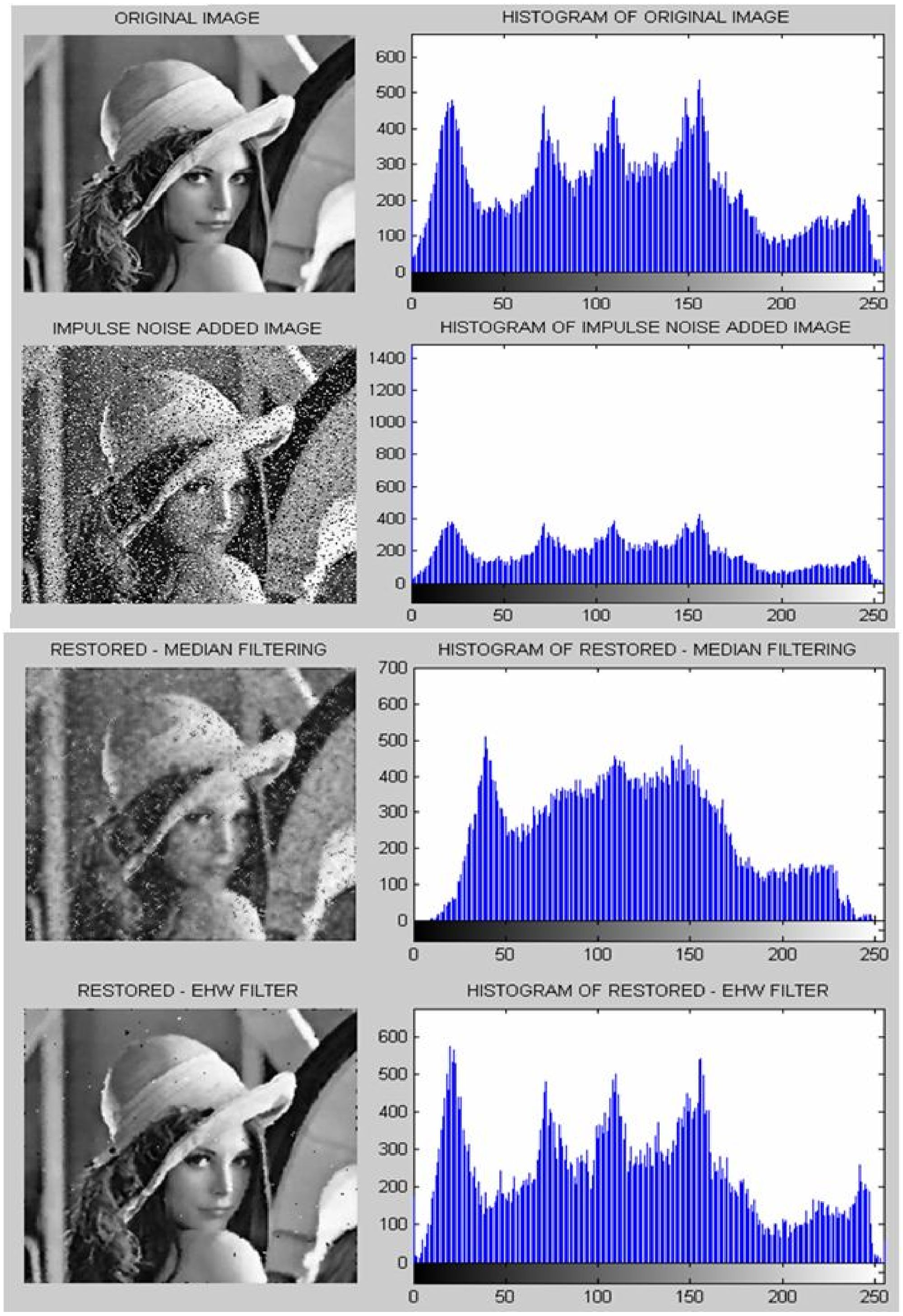


FIGURE 9 a: Original Image,
FIGURE 9 b: Salt and Pepper noise added image
FIGURE 9 c: Restored image using Sober filter
FIGURE 9 d: Restored image using EHW

The comparision results are given in Table 1. It is clear that EHW filter outperforms the other filters for both Gaussian and salt & pepper noise. The nonlinear part of EHW filter preserves the edges and removes the Gaussian noise effectively. The linear part smoothens the salt and pepper noise and removes the spurious part of the image.

Image	**PSNR (dB)**	**MDPP**	**MSE**
Gaussian Noise of SD= 0.05			
Median Filter	24.95	180606	23.08
EHW	27.05	143152	20.14
Salt and Pepper			
Sober Filter	24.17	173449	23.96
EHW	26.69	133399	19.44

TABLE 1: Comparison Study

9. CONSLUSION & FUTURE WORK

In this work, a novel virtual reconfigurable circuit based image filter was presented. It is shown that it is possible to successfully evolve noise removal filters that produce better image quality than a standard median and sober filter. The n-bit configuration information that determines the function of the functional block and the connection to the inputs is chosen optimally. Also, the design guarantees that, in this constrained structure, no random configuration information can ever destroy the chip. The fitness calculation part is the computationally time consuming operation and hence, in this work, this part is performed inside the FPGA along with the VRC. The utilized resources of the FPGA are also found to be practical and any commercially available FPGA can be used for implementing the EHW. If noise corrupting the image varies, a more accurate VRC filter is swapped into the FPGA hardware.

Future Work

To reduce the computational complexity and to enhance the speed, each kernel can be executed on a floating point processor. This can overcome the limitation of architectures which support only binary operations.

10. REFERENCES

1. Borgatti, M. Lertora, F. Foret, B. Cali, L. *" A reconfigurable system featuring dynamically extensible embedded microprocessor, FPGA, and customizable I/O"*. IEEE Journal of Solid-State Circuits, Vol.38(3): 521- 529, Mar 2003

2. Christian Haubelt, Thomas Schlichter, and Jurgen Teich, *"Improving Automatic Design Space Exploration by Integrating Symbolic Techniques into Multi-Objective Evolutionary Algorithms"*. International Journal of Computational Intelligence Research, Vol.2(3): 239–254, 2006

3. Guruva Reddy. A, Sri Rama Krishna. K, Giri Prasad. M.N and Chandra Bhushan Rao K, "Autonomously Restructured Fault tolerant image enhancement filter," ICGST International Journal on Graphics, Vision & Image Processing, Vol.08, issue-3, pp.35-40, Oct 2008.

4. Higuchi, T. Iwata, M. Keymeulen, D. Sakanashi, H. Murakawa, M. Kajitani, I. Takahashi, E. Toda, K. Salami, N. Kajihara, N. and Otsu, N, *"Real-world applications of analog and digital evolvable hardware"*. IEEE Transactions on Evolutionary Computation, Vol.3(3): 220-235, Sep 1999

5. Higuchi .T, N. Kajihara, *"Evolvable Hardware Chips for Industrial Applications"*. Communications of the ACM, Vol.42(4): 60-69, June 2006

6. Clark, G.R, San Diego and La Jolla. *"novel function-level EHW architecture within modern FPGAs"*. In Proceedings of Congress on Evolutionary Computation (CEC 99), California Univ, CA, 1999

7. Higuchi.T, Umezono and Tsukuba Ibaraki. *"Evolvable hardware with Genetic learning"*. In Proceedings of IEEE International Symposium on Circuits and Systems (ISCAS '96), 1996

8. Jie Li and Shitan Huang. *"Adaptive Salt-&-Pepper Noise Removal: A Function Level Evolution based Approach"*. In Proceedings of NASA/ESA Conference on Adaptive Hardware and Systems, June 2008

9. Kyrre Glette and Jim Torresen. *"A Flexible On-Chip Evolution System Implemented on a Xilinx Virtex-II Pro Device"*. In Proceedings of International conference on Evolvable systems, 2005

10. Sekanina.L. *"Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware"*. In Proceedings of Fifth International Conference on Evolvable Systems (ICES'03), 2003

11. Simon Harding. *"Evolution of Image Filters on Graphics Processor Units Using Cartesian Genetic Programming"*. In Proceedings of IEEE Congress on Evolutionary Computation, 2008

12. Goldberg, D. E. *"Genetic Algorithms in Search, Optimization & Machine Learning"*, Pearson Education, Inc, 1990.

13. Rafael C. Gonzalez and Richard E. Woods, *"Digital Image Processing"*, Second edition, Pearson Education, 2007