AN AREA-EFFICIENT ALTERNATIVE TO ADAPTIVE MEDIAN FILTERING IN FPGAS

Zdenek Vasicek and Lukas Sekanina

Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
email: vasicek@fit.vutbr.cz, sekanina@fit.vutbr.cz

ABSTRACT

This paper presents a new approach to the FPGA implementation of image filters which are utilized to remove the salt-and-pepper noise of high intensity (up to 70% of corrupted pixels). The proposed solution combines image filters designed by means of evolutionary algorithm with a simple human-designed preprocessing and post-processing unit. It provides the same filtering capability as a standard adaptive median filter; however, using four times less Virtex slices.

1. INTRODUCTION

This paper presents a new approach to the FPGA implementation of image filters which are utilized to remove the salt-and-pepper noise of high intensity (up to 70% of corrupted pixels). The goal is to show that by an innovative combination of evolved designs and conventional designs we are able to significantly reduce the overall implementation cost on a chip in comparison to standard approaches based on sophisticated filtering schemes, such as adaptive medians [1].

We will solely deal with the images corrupted by the shot noise, in particular *salt-and-pepper* noise in which the noisy pixels can take only the maximum or minimum values (i.e. 0 or 255 for 8-bit grayscale images). In most cases, this type of noise is caused by malfunctioning pixels in camera sensors, faulty memory locations in hardware, or errors in the data transmission.

Traditionally, the salt-and-pepper noise is removed by median filters. When the noise intensity is less than approx. 10% a simple median utilizing 3×3 or 5×5-pixel window is sufficient. However, it was shown that evolutionary design techniques are able to generate slightly better solutions than standard medians for this noise intensity [2] (see Fig. 5j–n). The images filtered by evolved filters are not so smudged and the area on the chip can by reduced by approx. 60%.

When the intensity of noise is increasing (10-90% pixels are corrupted), simple median filters are not sufficient and more advanced techniques have to be utilized. Among others, adaptive medians provide good results and simultaneously, their hardware implementation is straightforward [1]. However, as shown in Section 2, the problem is that these advanced techniques require a considerably larger area on a chip in comparison to simple median filters. The main reason is that larger filtering windows have to be applied and additional values (such as the maximum and minimum value of the filtering window) have to be calculated.

Unfortunately, the evolutionary design approach stated above which works up to 10% noise intensity does not work for higher noise intensities. It is shown in this paper that a *bank* of filters containing the evolved relatively simple filters which utilize only 3×3-pixel filtering windows is able to accomplish the same task as the adaptive median filter utilizing up to 7×7-pixel filtering windows; however, using a significantly reduced resources.

Section 2 surveys implementation costs of various FPGA implementations of image filters that can be used to suppress the salt-and-pepper noise. Section 3 introduces the basic idea of the proposed method. As the proposed method utilizes evolved image filters, Section 4 is devoted to the description of an evolutionary image filter generator which we use to routinely evolve (i.e. design) image filters. While the image filter generator is evaluated in Section 5.1, the proposed bank of filters is evaluated and compared to conventional solutions in Section 5.2. Obtained results are discussed in Section 6 and conclusions are given in Section 7.

2. THE COST OF CONVENTIONAL FILTERS

2.1. Standard Median Filters

Median-based non-linear filters play a prominent role among the filters utilized to suppress the salt-and-pepper noise [3, 4, 5, 6]. There are two approaches to the implementation of a standard median function: The first approach employs comparators which return the minimum or maximum value calculated from two input values. If the goal is to minimize the number of comparators, an optimal implementation exists for some problem instances. As we will use 3×3 and 5×5 filtering windows, the 9-input median (used also in Xilinx's reference implementation by Smith [7]) and the 25-input median [8] are important for us.

1-4244-1060-6/07/$25.00

# inputs	Number of slices optimal	SN bitonic	SN oe-merge	max. frequency
9 (3x3)	268	297	289	305 MHz
25 (5x5)	1506	1706	1582	305 MHz
49 (7x7)	unknown	4815	4426	305 MHz

Table 1. Results of synthesis for common median circuits.

The second approach is based on *sorting networks* (SN) which consist of compare-and-swap operations (also referred to as comparators) [9]. Sorting networks can be designed and optimized with the aim of reducing the number of comparators or delay. For a small number of inputs, the optimal (ad hoc) implementations are known. The comparator count and delay are considered as criteria for a given number of inputs. In order to design more complex SNs, general algorithms have been proposed. Among them, *bitonic sorting* and *odd-even merge sorting* provide the best results [9]. Then, effective implementations of n-input medians are obtained by pruning n-input SNs.

Table 1 compares the cost of pipelined implementations of mentioned median circuits. Note that in this paper, all results of synthesis are given for Virtex II Pro XC2vp50-7 FPGA which contains 23616 slices. We can observe that odd-even merge sorting requires fewer slices than bitonic sorting. Figure 5c shows that when the noise level is increasing, some details and edges of the original image are smeared by the median filter [10]. Although larger filtering windows allow removing more shots they produce considerably more smudged output images (see Fig. 5d, m).

2.2. Adaptive Median Filters

The adaptive median filter provides significantly better results than standard median filters especially for images corrupted with high noise intensity [1]. While many FPGA implementations exist for standard median filters, we have not found any FPGA implementation of the adaptive median filter in literature. Figure 1 shows the adaptive median filter we designed and implemented for purposes of comparison. The filter operates with a kernel of $S_{max} \times S_{max}$ pixels. Let S_{xy} denote the processed (i.e. central) pixel. The kernel is processed by a set of SNs with $3\times3, 5\times5, \ldots, S_{max} \times S_{max}$ inputs. Each SN provides the minimum, maximum and median value. Buffers are used to synchronize the outputs of all SNs and S_{xy}. The implementation is pipelined and tries to minimize the number of stages. Hence it is useful to utilize the odd-even merge SN which exhibits smaller delay. The output block (which is a simple combinational circuit) selects the output value according to the algorithm described in [1]. Table 2 compares the cost of adaptive median which utilizes up to 5×5 and 7×7 filtering window. The high visual quality of filtered images (see Fig. 5f, g) correlates with the high implementation cost.

Smax	SN bitonic # slices	max. freq	SN oe-merge # slices	max. freq	Latency [stages]
5x5	2220	305 MHz	2024	303 MHz	15
7x7	7297	302 MHz	6567	298 MHz	21

Table 2. Results of synthesis for adaptive median circuits.

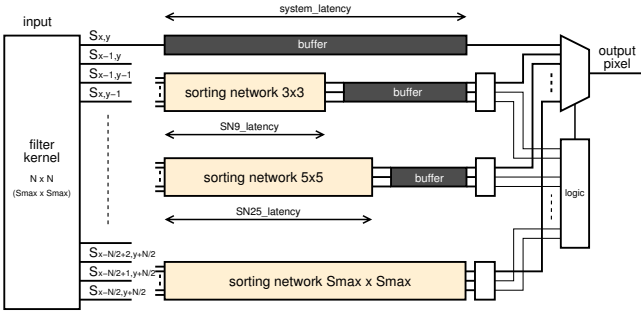

Fig. 1. HW implementation of adaptive median filter

2.3. Measures of the Visual Quality

The visual quality of filtered images is numerically expressed by the peak signal-to-noise ratio (PSNR) which is calculated as

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{MN}\sum_{i,j}(v(i,j) - w(i,j))^2}$$

where $N \times M$ is the size of image, v denotes the filtered image and w denotes the original image. Alternatively, the *mean difference per pixel* (mdpp) can be utilized

$$mdpp = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}|v(i,j) - w(i,j)|.$$

3. PROPOSED METHOD

In order to create a salt-and-pepper noise filter which generates filtered images of the same quality as an adaptive median filter and which is suitable for pipelined hardware implementation in FPGA, we propose to combine several simple image filters (utilizing the 3×3 window) that are designed by an evolutionary algorithm (EA). As Figure 2 shows the procedure has three steps:

(1) We analyzed various evolved filters reported in [2] and recognized that they have problems with the large dynamic range of corrupted pixels (0/255). A simple solution to this problem is to create a component which inverts all pixels with value 255, i.e. all shots are transformed to have a uniform value (i.e., 0). This is easy to implement in hardware by a comparator.

(2) The preprocessed image then enters a bank of n filters which operate in parallel. We selected n evolved filters which produce *different* results and which exhibit better-than-average filtering quality and utilized them in the bank.

Note that all these filters were designed by EA using the same type of noise and training image and with the same aim: to remove the 40% salt-and-pepper noise.

(3) Finally, the outputs coming from banks $1 \ldots n$ are combined by n-input median filter which can be easily implemented using comparators [9]. As the proposed system naturally forms a pipeline, the overall design can operate at the same frequency as a simple median filter.

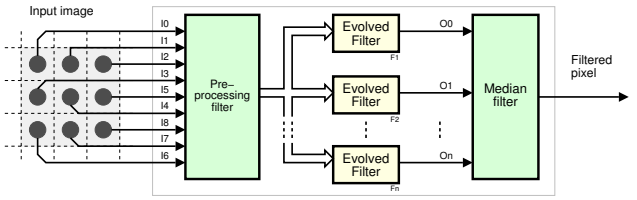

Fig. 2. A new architecture for salt-and-pepper noise removal

4. EVOLUTIONARY IMAGE FILTER GENERATOR

This section describes the system we employed to quickly design image filters which can be used in the bank of filters. The system consists of a genetic unit, array of reconfigurable elements and fitness calculation unit. The corrupted image is processed by a pipelined array of reconfigurable elements (called *Virtual Reconfigurable Circuit* (VRC) here [2]) whose configuration is generated by the genetic unit. While candidate filters are created and evaluated using a user logic available on the FPGA, the genetic operations are carried out in the PowerPC processor which is available as a hard core on the same FPGA. This type of evolvable hardware implementation was introduced in [11].

4.1. Image Filters in VRC

Every image operator is considered as a digital circuit of nine 8-bit inputs and a single 8-bit output, which processes gray-scaled (8-bit/pixel) images (see Fig. 3 top).

Fig. 3 also shows a corresponding VRC which consists of 2-input Configurable Logic Blocks (CFBs), denoted as E_i, placed in a grid of 8 columns and 4 rows. Any input of each CFB may be connected either to a primary circuit input or to the output of a CFB, which is placed anywhere in the preceding column. Any CFB can be programmed to implement one of functions given in Table 3. All these functions operate with 8-bit operands and produce 8-bit results. These functions were recognized as useful for this task in [2]. The reconfiguration is performed column by column. The computation is pipelined; a column of CFBs represents a stage of the pipeline. The configuration bitstream of VRC which is stored in a register array *conf_reg* consists of 384 bits. A single CFB is configured by 12 bits, 4 bits are used to select the connection of a single input, 4 bits are used to select one

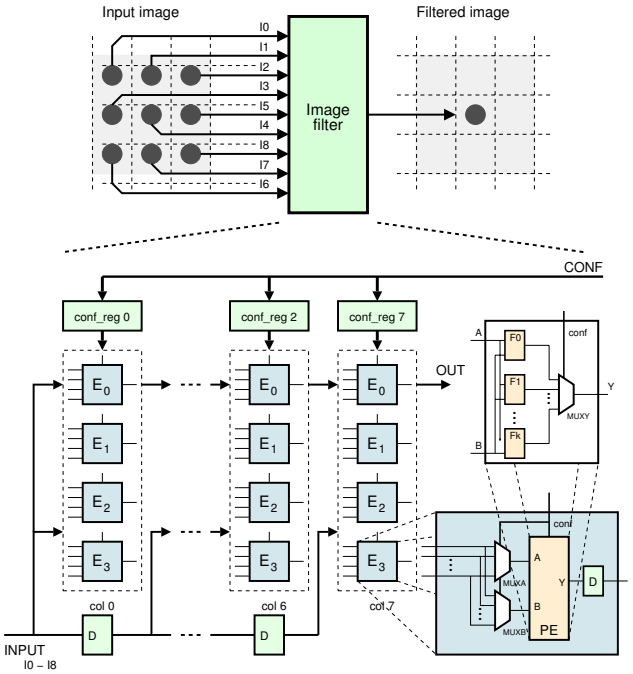

Fig. 3. Example of a candidate image filter representation by means of virtual reconfigurable circuit (bottom)

Table 3. List of functions implemented in each CFB

code	function	description	code	function	description
0	255	constant	8	$x \gg 1$	right shift by 1
1	x	identity	9	$x \gg 2$	right shift by 2
2	$255 - x$	inversion	A	$swap(x, y)$	swap nibbles
3	$x \vee y$	bitwise OR	B	$x + y$	$+$ (addition)
4	$\bar{x} \vee y$	bitwise $\bar{x}$ OR y	C	$x +^S y$	$+$ with saturation
5	$x \wedge y$	bitwise AND	D	$(x + y) \gg 1$	average
6	$\overline{x \wedge y}$	bitwise NAND	E	$max(x, y)$	maximum
7	$x \oplus y$	bitwise XOR	F	$min(x, y)$	minimum

of the 16 functions. Evolutionary algorithm directly operates with configurations of the VRC; simply, a configuration is considered as a chromosome.

4.2. Evolutionary Algorithm

The initial population of eight individuals is generated randomly. Then, two offspring are generated from each parent using a bit-mutation operator. A new population is selected from the eight parents and their sixteen offspring. We utilized a deterministic selection in which the eight-best scored individuals are selected as new parents. The evolutionary algorithm utilizes a single genetic operator — mutation, which is applied with the probability of 4.7-6.3% per bit. This mutation intensity was experimentally confirmed as the most suitable. No crossover operator is utilized in this type of EA [2].

4.3. Fitness Calculation

The fitness calculation is carried out by the Fitness Unit (FU). The corrupted image, original image and filtered image are stored in external SRAM memories which are accessible via memory controllers implemented in FPGA. The pixels of corrupted image u are loaded from external SRAM1 memory and forwarded to inputs of VRC. Pixels of filtered image v are sent back to the Fitness Unit, where they are compared with the pixels of original image w which is stored in another external memory, SRAM2. Filtered image is simultaneously stored into the third external memory, SRAM3. The design objective is to minimize the difference between the filtered image and the original image, i.e. the fitness value is calculated for $M \times N$-pixel image (note that border pixels are ignored) as

$$fitness = \sum_{i=1}^{M-2} \sum_{j=1}^{N-2} |v(i,j) - w(i,j)|.$$

4.4. Top Level Entity

As Fig. 4 shows, the proposed architecture is completely implemented in a single FPGA (except the SRAM memories). All components (except the VRC) are connected to the LocalBus.

In order to maximize the overall performance, the CU (Control Unit) plays the role of master and controls the entire system. In particular, it starts/stops the evolution, determines the number of generations and other parameters of search algorithm and generates control signals for the remaining components.

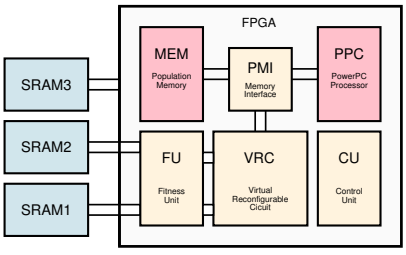

Fig. 4. A system for image filter evolution in FPGA

Upon request, the PowerPC generates a new candidate individual, i.e. it is idle in its main loop. Program memory of the PowerPC is implemented using on-chip Block RAM (BRAM) memories and connected to the LocalBus in order to send/read programs to/from an external PC which is connected with FPGA via a PCI bus. The population of candidate configurations is stored in on-chip BRAM memories. The population memory is divided into eight banks; each of them contains a single configuration bitstream of VRC. An additional bit (associated with every bank) determines data validity; only valid configurations can be evaluated. In order to overlap the evaluation of a candidate configuration with generating a new candidate configuration, at least two memory banks have to be utilized. While a circuit is evaluated, a new candidate configuration is generated. A new configuration is utilized immediately after completing the evaluation of the previous circuit.

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

The evaluation of candidate configurations is pipelined in such manner as there are no idle clock cycles. Therefore, time of evolution can be expressed as

$$t_{evol} = Q(M-2)(N-2)\frac{1}{f}$$

where Q is the number of evaluations, $N \times M$ is the number of pixels and f is the operation frequency.

Table 4. Results of synthesis of the image filter generator

VRC	IO blocks	BRAM	Slices	DFF
Available	852	232	23 616	49 788
4×8 CFBs	602	12	4 591	3 638
used	70%	5%	20%	7%

In order to implement the proposed evolutionary image filter generator, we used a COMBO6X card. Results of synthesis are summarized in Table 4. While the PowerPC works at 300 MHz, the logic supporting the PowerPC works at 150 MHz. The remaining FPGA logic (including VRC and FU) works at 50 MHz. Experimental results show that approximately 3,000 candidate filters can be evaluated per second ($N = M = 128$) which is 22 times faster than the same algorithm running at the Celeron@2.4GHz.

Table 5. Experimental results for the training Lena image (320,000 evaluations allowed in each run)

corrupted image	bits mutated	runs	mdpp			
			min	max	mean	std. dev.
5%-noise	18	64	0.333	3.450	2.010	1.240
10%-noise	24	349	0.828	7.390	2.650	2.190
20%-noise	20	139	0.870	12.10	2.680	1.330

5. EXPERIMENTAL RESULTS

5.1. Evaluation of the Image Filter Generator

This section illustrates main features of the implementation. The objective of this experiment is to evolve 3×3 salt-and-pepper noise filters. As a training image we utilized

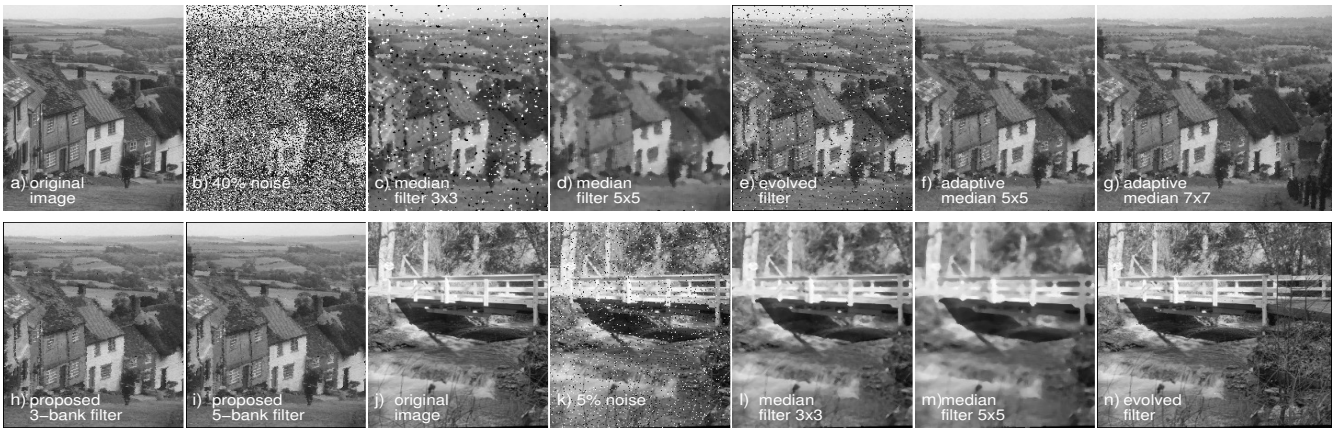

Fig. 5. Filtering the 40% noise (a–i) and filtering the 5% noise (j–n)

Table 6. Comparison of PSNR of the best-evolved filters and 3×3-median filter on a test set of 256×256-pixel images

test	5% noise		10% noise	
image	evolved	median	evolved	median
airplane	37.617	29.303	32.006	28.557
bird	42.692	38.242	36.675	36.990
bridge	35.522	26.040	30.699	25.662
camera	33.776	26.823	32.040	26.245
goldhill	37.503	27.927	32.776	27.524
lena	37.362	30.381	31.901	29.739

a 128×128-pixel version of Lena image which contains a given type of noise in some regions. Table 5 summarizes results of evolution for hundreds of independent runs. Table 6 presents results for a set of test images and compares evolved filters with a standard 3×3 median filter. Fig 5j–n shows an example—the bridge image.

5.2. Evaluation of the Bank of Filters

In order to evolve filters for the bank, a training 128×128-pixel image which was partially corrupted by 40% salt-and-pepper noise was utilized. Evolution was repeated 100 times; 1.5 million evaluations were performed in each run. According to the chromosomes of the five best-scored filters we created corresponding VHDL models and synthesized them. The first part of Table 7 shows that the implementation cost of evolved filters is much lower than the cost of 3×3 median circuit. Figure 6 shows three evolved filters.

The proposed approach and adaptive median filters are compared on several images of size 256×256 pixels which contain the salt-and-pepper noise with the intensity of 5%, 10%, 20%, 40%, 50% and 70% corrupted pixels. Table 8 summarizes results obtained for selected test images and two versions of the adaptive median filter and two versions of the bank filter (which contain the filters from Table 7). The higher PSNR, the better results.

Table 7. Result of synthesis for evolved filters utilized in the bank (filter1-5) and for the whole bank filters

filter	# slices	area	max. frequency	latency
filter1	156	0.7%	316 MHz	8
filter2	199	0.8%	318 MHz	8
filter3	137	0.6%	308 MHz	8
filter4	183	0.8%	321 MHz	8
filter5	148	0.6%	320 MHz	8
filter	**# slices**	**area**	**max. frequency**	**latency**
3-bank	500	2.1%	308 MHz	11
5-bank	843	3.6%	305 MHz	13

Surprisingly, only three filters utilized in the bank are needed to obtain a bank filter which produces images of at least comparable visual quality to the adaptive median filter. This fact is demonstrated by Figure 5f–i where the visual quality of the images filtered by the adaptive median and 3-bank filter is practically undistinguishable.

6. DISCUSSION

An obvious question is: How is it possible that three (five, respectively) filters evolved with the aim of removing 40%-salt-and-pepper noise are able to suppress the salt-and-pepper noise with the intensity up to 70%? Moreover, none of these filters does work sufficiently in the task which it was trained for (the 40% noise). Our explanation is that although these filters perform the same task, they operate in a *different* way. While a median filter gives as its output one of the pixels of the filtering window, evolved filters sometime produce new pixel values. By processing these n-values in the n-input median, the shot is suppressed. We tested several variants of evolved filters in the bank but never observed a significant degradation in the image quality.

The proposed approach was evaluated on a class of images which belong to the category of "home photo gallery". Future work will be devoted to testing the proposed filtering

Table 8. PSNR for adaptive median filters with the kernel size up to 7 × 7 and bank filters containing 3 and 5 filters

5 × 5	Adaptive median filters					
img/noise	**5%**	**10%**	**20%**	**40%**	**50%**	**70%**
goldhill	31.605	31.155	30.085	26.906	24.290	15.859
bridge	29.936	29.474	28.064	24.993	22.567	14.781
lena	34.434	33.665	31.210	27.171	24.433	15.468
pentagon	33.108	32.767	31.460	28.235	25.217	16.315
camera	30.862	30.367	28.560	25.145	22.675	14.973
7 × 7						
img/noise	**5%**	**10%**	**20%**	**40%**	**50%**	**70%**
goldhill	31.605	31.155	30.085	27.315	25.961	20.884
bridge	29.943	29.474	28.058	25.177	23.710	19.060
lena	34.425	33.655	31.207	27.529	25.984	20.455
pentagon	33.108	32.767	31.460	28.621	27.175	21.654
camera	30.862	30.367	28.560	25.298	23.852	19.242

3-bank	Bank filters					
img/noise	**5%**	**10%**	**20%**	**40%**	**50%**	**70%**
goldhill	36.614	33.759	30.619	27.716	25.867	19.091
bridge	34.064	31.458	28.992	25.830	24.282	18.333
lena	31.424	30.304	28.162	25.684	24.137	18.324
pentagon	37.445	34.631	31.890	28.681	26.577	18.437
camera	34.252	30.576	28.185	25.284	23.720	17.850
5-bank						
img/noise	**5%**	**10%**	**20%**	**40%**	**50%**	**70%**
goldhill	37.216	34.392	31.131	27.966	25.965	19.079
bridge	34.828	32.321	29.714	26.124	24.441	18.327
lena	31.448	30.393	28.424	25.881	24.203	18.314
pentagon	38.017	35.201	32.411	28.945	26.683	18.435
camera	34.622	31.091	28.740	25.576	23.919	17.845

scheme on other types of images. Anyway, results obtained for this class of images are quite promising from the application point of view. We can reach the quality of adaptive median filtering using a 3-bank filter; however *four times less* resources are utilized. This can potentially lead to a significant reduction of power consumption of a target system. Moreover, as the adaptive medians require larger filtering windows than bank filters they also require more logic to implement input FIFOs. For example, the adaptive median with kernel size up to 7 × 7 needs seven input FIFOs. Each FIFO stores the whole row of the filtered image. This overhead is *not* included in the implementation cost of filters in Tables 2 and 7. Note that the proposed filter bank, which can comprise an arbitrary number of evolutionary designed filters working in parallel, needs three input FIFOs only.

7. CONCLUSIONS

The proposed solution combines image filters designed by means of evolutionary algorithm with a simple human designed preprocessing and post-processing unit. It provides the same filtering capability as a standard adaptive median filter; however, using four times less slices. As a promising approach for future investigations, it seems to combine evolved designs with conventional designs. In future work, we would like to test the proposed scheme for other types of noise.

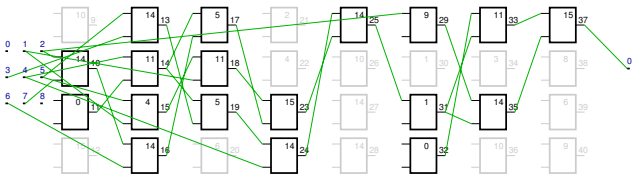

Fig. 6. Example of evolved filter for the 40% salt-and-pepper noise utilized in the 3-bank filter

Acknowledgements

This research was partially supported by the Grant Agency of the Czech Republic under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* and the Research Plan No. MSM 0021630528 *Security-Oriented Research in Information Technology*.

8. REFERENCES

[1] H. Hwang and R. A. Haddad, "New algorithms for adaptive median filters," in *Proc. SPIE Vol. 1606, Visual Communications and Image Processing '91*, Nov. 1991, pp. 400–407.

[2] L. Sekanina, *Evolvable components: From Theory to Hardware Implementations*, ser. Natural Computing. Springer-Verlag Berlin, 2004.

[3] N. H. Yung and A. H. Lai, "Novel filter algorithm for removing impulse noise in digital images," in *Proc. SPIE Vol. 2501, p. 210-220, Visual Communications and Image Processing '95, Lance T. Wu; Ed.*, L. T. Wu, Ed., Apr. 1995, pp. 210–220.

[4] L. Bar, N. Kiryati, and N. Sochen, "Image deblurring in the presence of salt-and-pepper noise," in *Scale Space*, 2005, pp. 107–118.

[5] M. Nikolova, "A variational approach to remove outliers and impulse noise," *J. Math. Imaging Vis.*, vol. 20, no. 1-2, pp. 99–120, 2004.

[6] S. A. Fahmy, P. Y. K. Cheung, and W. Luk, "Novel fpga-based implementation of median and weighted median filters for image processing." in *Proc. of the 2005 Int. Conf. on Field Programmable Logic and Applications (FPL)*. IEEE, 2005, pp. 142–147.

[7] J. L. Smith, "Implementing median filters in xc4000e fpgas," *XCell*, vol. 23, no. 1, p. 16, 1996.

[8] N. Devillard, "Fast Median Search: An ANSI C Implementation," 1998, http://ndevilla.free.fr/median/median.pdf.

[9] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.

[10] E. R. Dougherty and J. T. Astola, Eds., *Nonlinear Filters for Image Processing*, ser. SPIE/IEEE Series on Imaging Science & Engineering. SPIE/IEEE, 1999.

[11] Z. Vasicek and L. Sekanina, "An evolvable hardware system in Xilinx Virtex II Pro FPGA," *Int. J. Innovative Computing and Applications*, vol. 1, no. 1, pp. 63–73, 2007.