*International Research Journal of Innovations in Engineering and Technology (IRJIET)*

**ISSN (online): 2581-3048**
**Volume 6, Issue 12, pp 38-42, December-2022**
https://doi.org/10.47001/IRJIET/2022.612005

# Image Processing Algorithms Implemented in FPGAs

**Rustem Popa**

Department of Electronics and Telecommunications "Dunarea de Jos" University of Galati, Romania

***Abstract* - In this paper, we implemented two simple image processing algorithms in the MATLAB environment and then in the FPGA, and then we compared the results in terms of accuracy and execution time. The first algorithm deals with the pseudo-coloring of a monochrome image of 256 x 256 pixels, by assigning to each pixel in the original image three other values, corresponding to the RGB matrices in the colored image. The assignment of these values was based on a conversion table that generates 16 different colors in the HOT color scale. The second algorithm generates the negative image of the monochrome image by calculating the new values of the pixels making the difference between 255 and their initial value. For each algorithm discussed here, the images obtained in MATLAB R2014a are compared with those obtained in the Xilinx ISE 14.7 environment in terms of accuracy and execution speed.**

***Keywords:*** Image processing, Pseudo-coloring, HOT scale, Circuit simulation, Field Programmable Gate Arrays (FPGAs), MATLAB environment, Xilinx ISE.

## I. INTRODUCTION

As we know, algorithms are usually implemented in software and the programs built for this purpose run on standard hardware architecture. However, there are specific applications, in which the execution speed of the algorithm is essential, and image processing requires the existence of some hardware resources to ensure a maximum processing speed.

FPGA circuits offer two major advantages for achieving this goal: on the one hand, they have a very high degree of integration that provides the necessary hardware resources, and on the other hand, they are programmable by the user, and this allows the construction of highly efficient parallel hardware structures in signal processing, including images.

In the literature there are numerous papers dealing with image processing using FPGAs. A "versatile, modular and scalable platform for test and implementation of low-level image processing algorithms under real-time constraints" was presented in [1]. This implementation consists of a set of FPGAs organized in a systolic architecture. Although the circuits use a modest frequency of 66 MHz, the parallelism of the structure offers an impressive performance of about 6 GOPs with the possibility of extension to 9 GOPs. In [2], the authors implement several algorithms for processing binary images and compare their performance from the point of view of memory requirements and execution time. The paper [3] discussed the design of an image processing environment using the circuit structure of the XC6216, a well-known FPGA that allowed cell-level reconfiguration. The authors of the paper [4] introduce the concept of hardware skeletons, which are parameterized descriptions of some task-specific architecture, to which the user can introduce parameters such as values, functions, or even other skeletons. In [5] image processing is done using a hardware/software architecture based on hardware featuring a FPGA co-processor and a host computer which runs a LABVIEW application.

This paper presents two simple image processing algorithms, which were implemented in software, using the MATLAB environment, and then in FPGA. For each case, a comparison is made of the results in terms of accuracy and speed of the algorithm, using the same processed image.

For the implementation of the algorithms in the software, we used the MATLAB R2014a environment, and for the implementation in the FPGA we used the Xilinx ISE 14.7 environment.

This paper is organized as follows. Section II describes the implementation of the proposed algorithms in the MATLAB environment. The implementation of the algorithms in FPGA is done in Section III, using the ISE 14.7 environment from Xilinx and the Verilog Hardware Description Language. Section IV describes the main experimental results by comparing the images in terms of accuracy and execution speed. Finally, Section V concludes the paper.

## II. IMPLEMENTATION IN MATLAB

The image processed by the two algorithms is the well-known "Lena", which was downloaded from Internet, being used in the paper [6]. The image resolution is 256 x 256 pixels, and each pixel is represented by 8 bits, having values between 0 and 255. We used the original monochrome image, without using any other transformations. This image is represented in Figure 1.

 

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 6, Issue 12, pp 38-42, December-2022

https://doi.org/10.47001/IRJIET/2022.612005

**Figure 1: Monochrome image of "Lena" [6]**

### 2.1 Implementation of the pseudo-coloring algorithm

The 256 possible pixel values in the monochrome image are divided into 16 regions, each corresponding to a different color. The 16 pixel values in each region will get the same color. Each pixel in the original image will have a correspondence in 3 other values, each of 8 bits, for each color pixel in the matrices R, G and B, and then we concatenated the results, obtaining the color image [7].

In this paper we used the HOT color scale, in which the colors change smoothly from black, through shades of red, orange, and yellow, to white. The image colored by this method is represented in Figure 2. It is by no means identical to the original "Lena" color image, but our goal is to show that there is no difference between the image implemented in MATLAB and the one implemented in FPGA.

### 2.2 Implementation of the negative image

The pixel values in the monochrome image are between 0 and 255. The black color pixel has the value 0, and the white color has the maximum value of 255.

**Figure 2: Negative image in MATLAB and in FPGA**

**Figure 3: Colored image in MATLAB and in FPGA**

To obtain the negative image, we will subtract the pixel values from the maximum value of 255. The negative image thus obtained is represented in Figure 3.

The two algorithms were implemented in MATLAB R2014a environment on a computer with an Intel i3-6006U processor at 2 GHz. The pseudo-coloring algorithm uses "for" cycles to assign different values to groups of pixels located in the same color region, and the negative image generation algorithm uses a simple operation of subtracting two matrices. Therefore, we expect this algorithm to be faster than the first one. In fact, the running time for the pseudo-coloring algorithm is about 9000 µs, and for the negative image generation algorithm about 50 µs.

## III. IMPLEMENTATION IN FPGA

Figure 4 represents the structure of the project in FPGA, implemented in circuits from the Spartan 3E family and which was also used in the works [8] and [9].

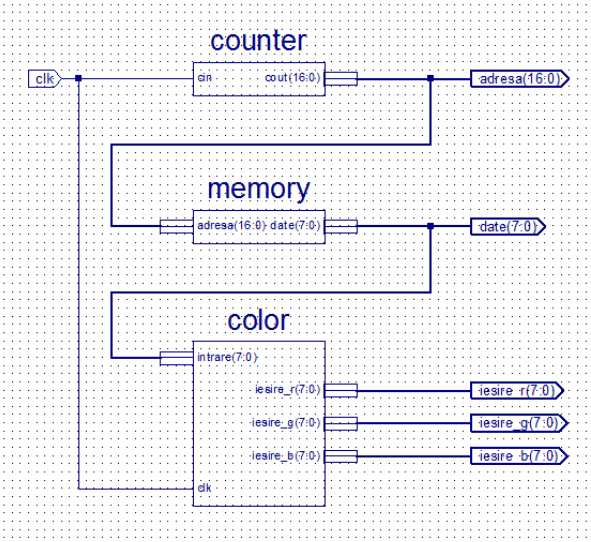


**Figure 4: Schematic diagram in Xilinx ISE 14.7**

 

International Research Journal of Innovations in Engineering and Technology (IRJIET)

ISSN (online): 2581-3048

Volume 6, Issue 12, pp 38-42, December-2022

https://doi.org/10.47001/IRJIET/2022.612005

The processed image is stored in the "memory" block, being read from the file using the instruction $readmemh. The format of the file containing the image is the HEX format, obtained by converting the JPG format in which the original image was provided ([10]).

The image has 256 x 256 pixels, that is, it contains a total number of 65536 pixels, each pixel being represented by an 8-bit number. To store this image, we chose a memory with 17 address lines (numbered from 16 to 0) and 8 bits of output data (numbered from 7 to 0). The "counter" block is a counter that increments the address lines of the memory. The clock signal is the one that changes the addresses and each pixel in the image is read and processed during a clock period.

The third block in the diagram is the one that contains the desired algorithm. The "color" block implements the pseudo-coloring algorithm. This block has an 8-bit input for each pixel in the original image and three 8-bit outputs, corresponding to the pixels in the 3 color matrices R, G and B.

### 3.1 Implementation of the pseudo-coloring algorithm

In the following we present a fragment of the Verilog code for the "color" block. The values assigned to the output variables are taken from the correspondence table for the HOT color scale.

```
module color(
    input clk,
    input [7:0] intrare,
    output reg [7:0] iesire_r, iesire_g, iesire_b
    );

    always @(negedge clk)
        begin
            if ((intrare > 16) && (intrare < 33))
                begin
                    iesire_r = 77; iesire_g = 11; iesire_b = 57;
                end
            else if ((intrare > 32) && (intrare < 49))
                begin
                    iesire_r = 102; iesire_g = 31; iesire_b = 73;
                end
            ..........................................
        end
endmodule
```

The verification of the proposed project is done by simulation. For this purpose, a Verilog file is generated for testing, and the simulation generates the 3 files containing the color matrices. Function $fopen opens 3 files in HEX format, and the data obtained at the 3 outputs are saved with the function $fwrite. The data were saved in binary format, the values of the image pixels being in text format, that is, for each bit of 1 or 0, the ASCII code of the respective character is stored. A simple program can convert this information into 8 bits samples. The image obtained in this way is identical to the image obtained in MATLAB, which was represented in Figure 2 (the PSNR between the two images is infinite).

### 3.2 Implementation of the negative image

To implement this algorithm, we used the same scheme as in Figure 4, but the "color" block is now replaced by another similar block, only this new block also has an 8-bit input and a single 8-bit output, which provides the resulting pixels of the negative image.

The complete Verilog code for this module is shown below. Only one operation is performed, namely the difference between two matrices. Thus, a maximum parallelism is ensured for the maximum efficiency of the algorithm from the point of view of execution speed.

```
module color(
    input clk,
    input [7:0] intrare,
    output reg [7:0] iesire
    );

    always @(negedge clk)
        begin
            iesire = 255 - intrare;
        end
endmodule
```

The test file used in the simulation is similar to the one used for testing the pseudo-coloring algorithm. The same functions $fopen and $fwrite are used, except that now a single file is generated. And this time, the two images generated in MATLAB and in FPGA, which are represented in Figure 3, are identical.

## IV. RESULTS AND DISCUSSIONS

The waveforms shown in Figure 5 represent the simulation results for the first pixels in the image in the case of the pseudo-coloring algorithm. We notice that in the first clock period, the value of the first pixel in the monochrome image is 00101010 in base 2, that is 42 in decimal. The values of the three RGB outputs are: output_r = 01100110, that is 102 in decimal, output_g = 00011111 , that is 31 in decimal, and output_b = 01001001, that is 73 in decimal. These values are exactly the ones in the conversion table, which were entered in the Verilog code that describes the "color" block discussed earlier (as we can see in Verilog code, the number 42 is between 32 and 49).

HDL Synthesis Report suggests the necessary hardware resources for the implementation of the pseudo-coloring algorithm. A 16-bit counter, three 8-bit registers, 28 comparators and a 512 kb memory, are required.

 

*International Research Journal of Innovations in Engineering and Technology (IRJIET)*

**ISSN (online): 2581-3048**

**Volume 6, Issue 12, pp 38-42, December-2022**

https://doi.org/10.47001/IRJIET/2022.612005

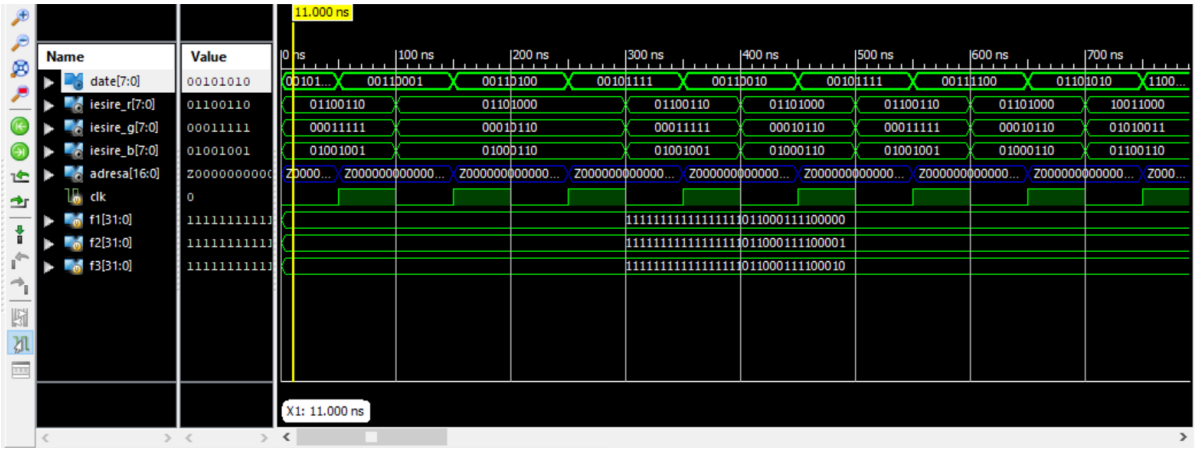


**Figure 5: R, G, and B outputs for the first eight pixels of the colored image**

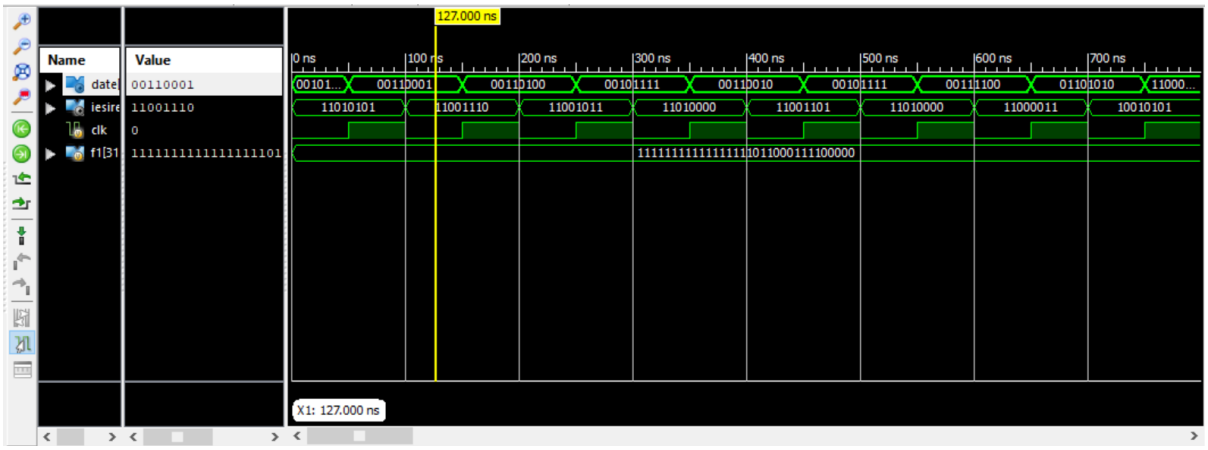


**Figure 6: Output for the first eight pixels of the negative image**

We can choose, for example, the FPGA circuit XC3S1600E, which contains 1600k equivalent logic gates and a RAM memory of 648 kb.

We saw that the execution time of the pseudo-coloring algorithm in MATLAB is about 9000 µs. The frequency of the clock signal for the circuit XC3S1600E is 300 MHz. If each image pixel is processed in one clock period, then the total image coloring time is 218 µs (i.e. 65536 pixels x 3,33 ns/pixel). So, the execution of this algorithm is about 40 times faster in hardware than in software.

Figure 6 represents the waveforms for the first eight pixels of the negative image. As we can see, the binary value of the first pixel in the image is 00101010 in base 2, that is 42 in decimal. The value of the processed pixel is the base 2's complement, i.e. 11010101, which is 213, exactly the value obtained by subtracting the number 42 from 255. Likewise, for the value of the next pixel 00110001, the negative value of this pixel is 11001110 and so on.

The execution time in MATLAB is about 50 µs, and the execution time in the circuit XC3S1600E is the same as the one from the previous algorithm (each bit is processed in a clock period), that is 218 µs.

## V. CONCLUSION

In this paper we implemented two simple algorithms in MATLAB and then in FPGA. The resulting images for a given algorithm are identical for the two implementations because the pixel values are integers and there are no rounding errors typical of floating point processing.

As is known, implementation in hardware is usually faster than implementation in software and this is confirmed by the first algorithm, which has a 40 times faster execution speed in hardware. In the pseudo-coloring algorithm, each pixel is processed for a clock period, both in software and in hardware, so the algorithms are identical from a constructive point of view.

The results are different in the case of the second algorithm, due to the fact that its implementation in MATLAB is vectorial, while in FPGA it remained the same as in the first case, in which each pixel in the image changes in a period of clock.

 

ISSN (online): 2581-3048
Volume 6, Issue 12, pp 38-42, December-2022
https://doi.org/10.47001/IRJIET/2022.612005

A simple algorithm can be directly implemented in the FPGA, using an HDL language, such as Verilog HDL, as we did in this experiment. More complicated algorithms can be transformed from MATLAB or SIMULINK code into HDL code, using HDL coder from MATLAB. FPGA circuits provide very good support for future implementations of more complex algorithms in hardware.

## REFERENCES

[1] G. Saldaña-González, and M. Arias-Estrada, "FPGA Based Acceleration for Image Processing Applications", in the book: "*Image Processing*", INTECH, Croatia, pp. 477-492, 2009

[2] A. Nieto, V. M. Brea, and D. L. Vilariño, "An FPGA-based Topographic Computer for Binary Image Processing", in the book: "*Image Processing*", INTECH, Croatia, pp. 493-516, 2009

[3] A. Bouridane, D. Crookes, P. Donachy, K. Alotaibi, and K. Benkrid, "A high level FPGA-based abstract machine for image processing", *Journal of Systems Architecture*, vol. 45, pp. 809-824, 1999

[4] K. Benkrid, D. Crookes, and A. Benkrid, "Towards a general framework for FPGA based image processing using hardware skeletons", *Parallel Computing*, vol. 28, pp. 1141–1154, 2002

[5] J.A. Kalomiros and J. Lygouras, "Design and evaluation of a hardware/software FPGA-based system for fast image processing", *Microprocessors and Microsystems*, 2008, doi: 10.1016/j.micpro. 2007.09.001

[6] T. Chen, *et al.* "Combined Digital Signature and Digital Watermark Scheme for Image Authentication", *Info-tech and Info-net, 2001. Proceedings, vol.5, ICII 2001*, Beijing, (2001) https://www.researchgate.net/publication/3935609

[7] R. C. Gonzalez, R. E. Woods, and S. L. Eddins, "Digital Image Processing using MATLAB", 2nd ed., *Gatesmark Publishing, ch.7*, pp. 318-376, 2009

[8] R. Popa, "ECG Signal Filtering in FPGA", *The 6-th International Symposium on Electrical and Electronics Engineering, ISEEE 2019, Galați, Romania*, 18-20 October 2019, (in IEEE Explore) https://ieeexplore.ieee.org/document/9136119/

[9] M. S. Pavel, R. Popa, "An Algorithm for Pseudocoloring Images in FPGA", *The 7-th International Symposium on Electrical and Electronics Engineering, ISEEE 2021, Galați, Romania*, 28-30 October 2021, (in IEEE Explore) https://ieeexplore.ieee.org/document/9628810/

[10] Van Loi Le, FPGA4Student. Site with Verilog/ VHDL Projects, 2016, URL: https://www.fpga4student.com/2016/11/image-processing-on-fpga-verilog.html

## AUTHOR'S BIOGRAPHY

**Rustem Popa** received the B.S. degree in electronics engineering from Bucharest Politehnica University in 1984, and Ph.D. degree at "Dunarea de Jos" University in Galati, Romania, in 1999. His research interests include digital electronics, medical electronics and soft computing. He is the author and co-author of 7 books and over 60 journal and conferences papers.

**Citation of this Article:**

Rustem Popa, "Image Processing Algorithms Implemented in FPGAs" Published in *International Research Journal of Innovations in Engineering and Technology - IRJIET*, Volume 6, Issue 12, pp 38-42, December 2022. Article DOI https://doi.org/10.47001/IRJIET/2022.612005

*******

 