Acta Scientific COMPUTER SCIENCES

Volume 4 Issue 1 January 2022

Research Article

How to Implement an Algorithm in Hardware

Rustem Popa*

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

***Corresponding Author:** Rustem Popa, Department of Electronics and Telecommunications, "Dunarea de Jos" University in Galati, Romania.

Received: November 23, 2021

Published: December 23, 2021

Abstract

After a brief review of the possibilities of implementing an algorithm, we analyze the implementation of an algorithm for coloring a monochrome image in an FPGA (Field Programmable Gate Array) circuit. Each pixel in shades of gray in the monochrome image receives one of 16 possible colors, according to its position in the color image. Although the approximation is crude, given that the original image has more than 16 million distinct colors, a comparison between the two images indicates a PSNR (Peak Signal to Noise Ratio) value of almost 25 dB, and the appearance of the color image is acceptable. The algorithm was implemented in the MATLAB environment and then in the FPGA using the integrated Xilinx ISE design environment. The color images generated in this way were identical, but the execution speed in FPGA was 185 times faster.

Keywords: Image Processing; Image Coloring; Circuit Simulation; Field Programmable Gate Arrays (FPGAs); MATLAB Environment

Introduction

When developing or testing an algorithm we usually write a program in an appropriate programming language and then test it on a general purpose computer. Execution time is usually not critical, but it can be substantially reduced by using parallel programming techniques. So the algorithm is implemented in the software and is executed on a general hardware structure, such as a PC. If the complexity of the algorithm is not very high, then we can use a classic microcontroller structure, and the program instructions are stored in an internal ROM memory of the microcontroller. Another option is to implement the algorithm in a hardware structure adapted to run the algorithm as efficiently as possible. There are many algorithms, based for example on artificial intelligence, which are implemented directly in ASIC (Application Specific Integrated Circuit) integrated circuits (neural networks, fuzzy logic, etc.).

However, to test a prototype, before manufacturing it in large series on an ASIC, an FPGA circuit is usually used. In this case, the algorithm is implemented with maximum efficiency, by building a parallel hardware structure using the internal resources available in the FPGA. In this paper our goal is to implement in FPGA a simple algorithm for coloring a monochrome image and then to verify the operation of the algorithm by comparison with its implementation in MATLAB. There are numerous examples in the literature of highly complex algorithms implemented in FPGAs, such as an evolutionary image filtering algorithm presented in [1] or the PSO (Particle Swarm Optimization) algorithm presented in [2].

Implementation of the algorithm in MATLAB

We chose to process a well-known image with a resolution of 256 x 256 pixels, each pixel being represented on 8 bits. The "Lena" color image we used was downloaded from the Internet, being used in the paper [3], and the equivalent monochrome image was

used in [4] and downloaded from that location. In this example we used a nonlinear mapping, in which 16 distinct color steps are generated, based on the Table 1. For each monochrome pixel, which has a value between 0 and 255, we assigned another 3 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 [5].

Figure 1: Color image of "Lena" [3].

Figure 2: Monochrome image of "Lena" [4].

For example, the value of the first pixel in the monochrome image is 42, then we assign to the corresponding pixels in the 3 matrices R, G, and B the values 102, 31, and 73 respectively and so on. The result of coloring the monochrome image using 16 color steps can be seen in figure 3.

Figure 3: Colored image in MATLAB.

Gray level	Values in matrices R, G, B		
	R	G	B
0 - 15	107	49	97
16 - 31	77	11	57
32 - 47	102	31	73
48 - 63	104	22	70
64 - 79	123	72	128
80 - 95	110	44	90
96 - 111	152	83	102
112 - 127	203	80	85
128 - 143	220	107	99
144 - 159	242	132	115
160 - 175	235	137	138
176 - 191	221	178	169
192 - 207	239	183	170
208 - 223	235	211	183
224 - 239	255	224	181
240 - 255	255	255	200

Table 1: Conversion Table.

Implementation of the algorithm in FPGA

We used a project structure similar to the one used in [6,7]. Schematics diagram of the project is represented in figure 4. The

pixel values of the monochrome image are read from a file and stored in the memory block entitled "memory". As shown in [8], the file containing the image information must be in HEX format. The transformation from JPG format to HEX format was done in MATLAB. The image has 256 x 256 pixels, which is a total number of 65536 pixels, each of them represented on 8 bits. We choose a memory with 16 address lines (numbered from 15 to 0 in figure 4) and 8 bits of output data (numbered from 7 to 0). A "counter" block generates the addresses applied to the memory and each pixel in image is read and processed during a clock period. So, for all 65536 pixels in the image we need the same number of clock periods. The coloring algorithm as was described above is implemented in the "color" block.

We simulate the operation of the circuit using a Verilog file for testing, which generates 3 files that contain the 3 color matrices. These output files may be in any convenient format: binary, decimal or hexadecimal. We preferred the binary format, because the information in these files is in text format, that is, for each bit of 1 or 0 we have the ASCII code of that character. A simple program can convert this information into 8 bits samples. In figure 5 we represented the colored image generated in FPGA, an image absolutely identical to the one generated in MATLAB (PSNR is infinite).

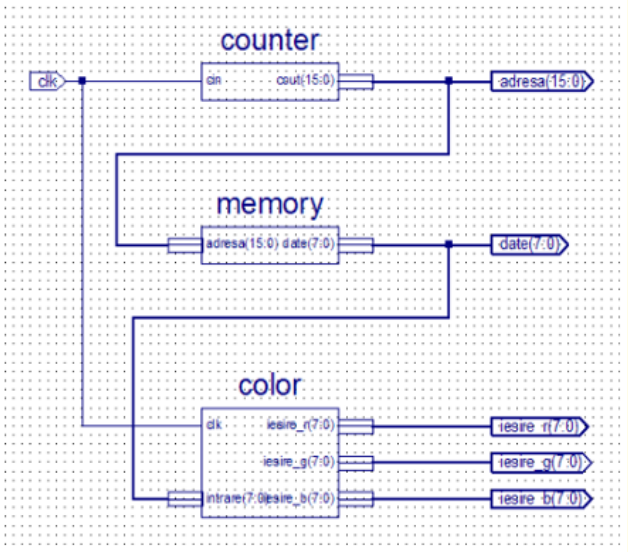

Figure 4: Schematics diagram of the project.

Figure 5: Colored image in FPGA.

Results and Discussion

If we compare the two images in figures 1 and 5 we find that there are differences. The initial image can have over 16 million colors (256 x 256 x 256), resulting from all possible combinations of values for matrices R, G, and B. We used only 16 colors to assign, according to table 1, one color for each group of gray pixels with similar values. There are also pixels that have erroneous colors, we notice that some brown pixels in the original figure appear in purple on the colored image. If we compare the two images, a PSNR of 24.15 dB is obtained. For more color levels, then the errors would have been smaller. The waveforms in the figure 6 represent the result of the simulation for the first pixels in the image. The value of the first pixel is 00101010 in base 2, that is 2A in hexadecimal or 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. So, we got exactly the values written in the "color.v" file for pixels with values between 32 and 47, as we can see in table 1.

The hardware resources required to implement the project are listed in the HDL Synthesis Report. We need a 16-bit up counter, three 8-bit registers, 28 comparators, and a memory of 65536 x 8 bits, that is 512 kb. As an example, we can use the XC3S1600E circuit, which contains about 1600k equivalent logic gates and a RAM memory of 648 kb.

We used the ISE Design Suite 14.1 from Xilinx and the MATLAB R2014a environment on a computer with an Intel i3-6006U processor at 2 GHz. If we consider a modest clock of about 100 MHz for FPGA, the total coloring time of the image is 655 µs (i.e. 65536 pixels x 10 ns/pixel). The execution time in MATLAB for the same algorithm is 121.19 ms. So, the execution of this algorithm is 185 times faster in hardware than in software.

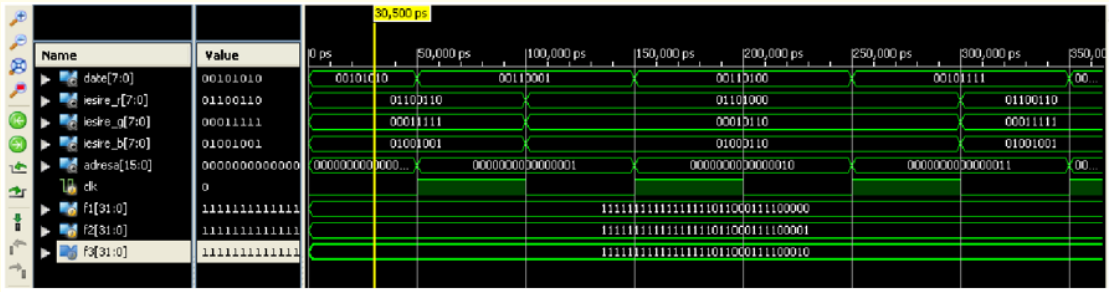

Figure 6: R, G, and B outputs for the first four pixels of the image.

Conclusion

In this paper we have considered a very simple algorithm that has been implemented in an FPGA. Following the same steps, more complicated algorithms can be implemented, provided that the existing hardware resources in the FPGA are sufficient. In our example, the existence of a large enough memory is essential for image storage, many smaller FPGA circuits do not have sufficient memory resources. The advantages of implementing algorithms in FPGA circuits are due to the optimization of available hardware resources and obtaining speeds tens of times higher than equivalent software implementations. If the algorithm is simple enough, then it can be implemented directly in an HDL language, such as the Verilog language, as we did in this experiment. All three component blocks of the project were generated using Verilog code, and then the graphic symbols were generated and they were interconnected as shown in the schematics diagram in figure 4. For more complicated algorithms there is also the option of automatic generation of HDL code (Hardware Description Language) using HDL Coder from MATLAB. FPGA circuits provide very good support for future implementations of highly complex algorithms in hardware.

Conflict of Interest

The author does not have conflicts of interests.

Bibliography

1. R Dobai and L Sekanina. "Image Filter Evolution on the Xilinx Zynq Platform". 2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013), IEEE Xplore (2013).
2. Y Maeda and N Matsushita. "Simultaneous Perturbation Particle Swarm Optimization and Its FPGA Implementation". in: A. Lazinica (ed.) Particle Swarm Optimization, In Tech, Croatia (2009): 347-362.
3. G Grammatikopoulos., *et al*. "Simple Matlab Tool for Automated Malignant Melanoma Diagnosis". *WSEAS Transactions on Information Science and Applications*, March (2007).
4. T Chen., *et al*. "Combined Digital Signature and Digital Watermark Scheme for Image Authentication". Info-tech and Info-net, 2001. Proceedings. Beijing 5 (2001).
5. R C Gonzalez., *et al*. "Digital Image Processing using MATLAB". 2nd ed., Gatesmark Publishing, ch. 7 (2009): 318-376.
6. R Popa. "ECG Signal Filtering in FPGA". The 6th International Symposium on Electrical and Electronics Engineering, ISEEE 2019, Galați, Romania, 18-20 October 2019, (2019).

7. M S Pavel and R Popa. "An Algorithm for Pseudocoloring Images in FPGA". The 7th International Symposium on Electrical and Electronics Engineering, ISEEE 2021, Galați, Romania, 28-30 October 2021, pp. 8 in Abstract Volume (paper 26) (2021).

8. Van Loi Le. "FPGA4Student". Site with Verilog/VHDL Projects, (2016).