1. Riyadh Zaghlool Mahmood[1], 2. Hiba-Aallah Tariq Abdullah[2]

University of Mosul (1), Northern Technical University (2)
ORCID: 1. orcid.org/0000-0001-7899-6475; 2. orcid.org/0000-0001-6276-0011

doi:10.15199/48.2023.01.23

FPGA-Based high speed two ways parallel histogram computation for grey image

Abstract *- In this paper approaches to the parallel architecture for local parallel histogram computation is studied. In this method, has been used many number of block RAM in FPGA based, each of them to perform a specific function must use a dual-ported of BRAM memory. These hardware techniques need one array of image and another one array for histogram. To reduce number of cycles in the FPGA implementation of our proposed technique read two operation memories at the same time*.

Streszczenie*. W tym artykule badane są podejścia do architektury równoległej do obliczania lokalnego histogramu równoległego. W metodzie tej wykorzystano wiele bloków pamięci RAM w układzie FPGA, każdy z nich do wykonywania określonej funkcji musi wykorzystywać dwuportową pamięć BRAM. Te techniki sprzętowe wymagają jednej tablicy obrazu i drugiej tablicy dla histogramu. Aby zredukować liczbę cykli w implementacji FPGA proponowanej przez nas techniki odczytujemy jednocześnie dwie pamięci operacyjne*. (***Oparte na FPGA szybkie, dwukierunkowe, równoległe obliczanie histogramu dla szarego obrazu***)

Keywords: FPGA, Histogram, Parallel, RAM
Słowa kluczowe: FPGA, histogram, obraz

1. Introduction

A histogram is a graphical diagram that shows the number of pixels of a video frame or digital image Have a contain intensity. It has one of image processing application [1]. Histogram extracting features from images that is simple and its features are fixed to image rotation. Histogram plots the number of an image pixels' distribution for each tonal value. The histogram for a dark image will have generality of its data points on the center and left side of the image, otherwise the histogram for a bright image with some shadows or dark places will have generality of its data points on the center and right side of the image. Finally, improvements in picture brightness and contrast can thus be obtained. The size of the histogram array is dependent on how many bits per pixel (bpp). If the bpp = n, 2^n elements exist in the histogram array that implements a flexible hardware architecture that is able to compute a range of histogram-based descriptor variants has several advantages. It can be modified with completely resynthesizing and reprogramming the FPGA [2]. The introduction of High-Level Synthesis (HLS) techniques and tools, as well as new features in high-end FPGAs including multi-port memory interfaces, has enabled designers to use FPGAs for memory-bound tasks along with compute-bound tasks. This article explains how to parallelize histogram as a memory-bound task using the OpenCL framework running on FPGA [3] efficiently. Parallel histogram computing necessitates a thorough rewrite of the method. In this context, this research eliminates memory access conflicts from histogram computing, allowing for a more flexible and effective parallelism utilization. While the planned pipeline architecture supports the streaming paradigm (each FSM state may be performed in only one clock cycle), it causes a memory access conflict that must be resolved [4]. To calculate the histogram of an image, an architecture is presented. This design achieves parallelism but requires sufficient resources and provides higher performance than prior serial techniques. This is one of the better methods for histogram computation in FPGA if resources are not a problem (Field Programmable Gate Array). Alternative ways are presented too to use the same design with fewer resources, resulting in a performance reduction [9]. In computer vision systems, image feature separation is a key stage in image segmentation. Unsupervised clustering of the generated data set can be one efficient and powerful way, although it is a computationally costly operation. A high-performance architecture for unsupervised data clustering is presented in this study [5.] Contextual Contrast Limited Adaptive Histogram Equalization (C-CLAHE) is a useful technique for reducing the noise amplification impact of adaptive histogram equalization (AHE) and improving picture visibility of local features. Despite the fact that C-CLAHE uses less memory than CLAHE, the interpolation method' complexity significantly increases the compute requirement [6].

Literature Review

In 2006, Orlando J. Hernandez, Member, provides a special-purpose VLSI architecture with high performance for the high-dimensional feature clustering in real time where data from images or video frames is taken. The design has been costed in hardware and prototyped in an FPGA environment. [5].

In 2007, Burak Unal, Ali Akoglu, implemented CLAHE is processed in real time thanks to the use of a large memory bandwidth. We present a method for implementing Contextual CLAHE via real-time interpolation. This is the first interpolation-based CLAHE implementation that is done on the FPGA, to our knowledge [6].

Module parallelism was proposed by Ernest Jamro, Maciej Wielgosz, and Kazimierz Wiatr in 2007, when a single module could take multiple input data in a single clock cycle. The fundamental disadvantage arises from this design. One is that the number of BRAMs which is required in total for parallel histogram calculations (or LUT programming) and the reading of the histogram is the product of the two (LUT conversion). As a result, the level of parallelism inside modules is quite limited [7].

In 2007, Shahbahrami, Ben Juurlink, and Stamatis Vassiliadis presented a method for counting the number of identical sub-words within a media register using parallel comparators. The numbers are then summed to the histogram array's values at the same time. Experiments using the Simple Scalar toolset reveal that the proposed solutions boost performance by a factor of 7.37 and 5.52, respectively, when compared to the fastest scalar version [8].

In 2016, Krishna Swaroop and Gautam Uurmi introduced a novel architecture for calculating the histogram in FPGA using 256-ways calculation and the generalized form of it, the n-ways calculation of the histogram. The presented architectures, both the n-ways or the 256-ways,

are superior compared to the old technique in regards of performance. Almost by 256 or n times, However, they use 256 or n times more resources, according to the analysis and results [9].

3. Histogram technique

At the first glance the histogram Treatment starts from (0 to 255) depending on the gray levels, (0) represents the dark area or blackness in the picture, while the number (255) represents the white color in the picture, and between two numbers are the gray gradations between the two colors [10][11]. A memory Block RAM (BRAM) needed to store the values of the histogram, where each value of the image will be the index of the histogram memory. Storing input image (128 * 128) pixels gray level needed a BRAM. The size of this BRAM is 16K bit, this is equivalent to 16,384 memory addresses for an image [12].

4. Hardware implementation of histogram calculation

First, the image is read via MATLAB program and converted into a gray-level image, and then the image is converted into an array of numbers and stored as a text file with the extension. coe, the next step is to call the .coe-extension text file. Store it in FPGA memory.

As illustrated in Fig. 1, the calculation module is made up of (BRAM) memory and incremental logic. The following shows how the module works: The BRAM is addressed by the input data (of which the histogram is being analyzed). The BRAM data out displays the number of previous occurrences of Data in at the BRAM address bus. The count is then increased by one and written back to the BRAM at the same location; and so forth.

The block diagram below shown in fig. 1 indicates to the mechanism for calculating the histogram in a serial method with a consumption of three clock cycles per pixel

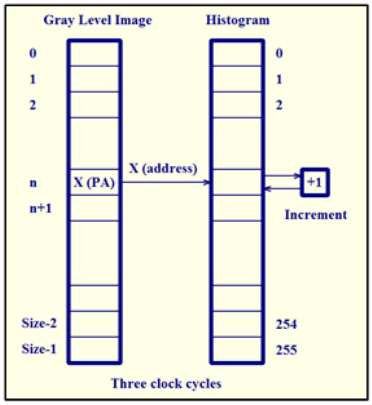


Fig. 1: Serial Method Histogram

5. Parallelizing Histogram Implementation Details

Memory conflicts must be avoided by the implementing hardware. Our presented concept makes use of a dual-ported memory which is then divided into two phases, as shown in Fig. 2 above. Two histogram arrays are computed in parallel in the first phase, one is for the pixels with addresses that are even-numbered. The other phase is for the addresses that are odd-numbered. The stored values in one memory of array addresses are utilized in the form of indices to the histogram array, according to the histogram functions. The pixel values can be read through both ports dout1A and dout1B in the first part of the cycle, as shown in Fig. 2. The incremented values of the histogram elements (updated values) are stored in the second part of the cycle. If the values of even and odd location in the image are different, then the value stored in histogram array indexed with the content of the odd location of image is incremented and content of the even location of image is incremented too, while if the values of even and odd location in the image are same, then the value stored in histogram array indexed with the content odd location of image is incremented by 2, while the even location is ignored.

The histograms arrays, which include the histogram values of even and odd numbered addresses, are combined in second phase. The final output is saved in a single histogram array.

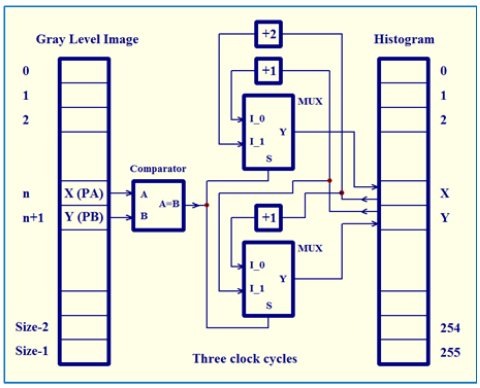


Fig. 2: Parallel method histogram

The execution time between Fig. 1 and Fig. 2 is difference. First figure takes 16384 clocks otherwise the second figure is consumes half of the previous time.

Fig. 3 illustrates the flowchart of serial histogram computation, while Fig. 4 shows the flowchart of parallel histogram computation.

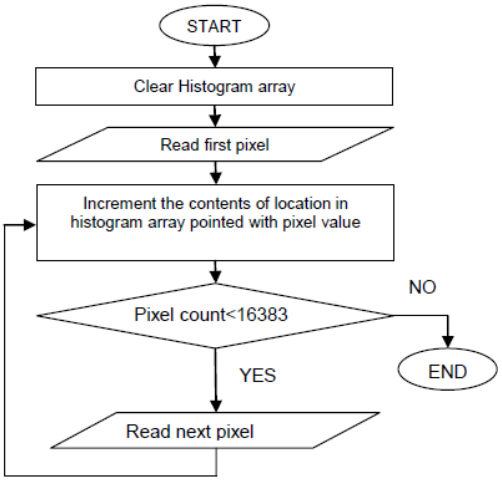


Fig. 3: serial histogram computation

Table 1: Execution time using MATLAB, Serial and sequential using FPGA with and without drawing processes with speedup

	Execution time in MATLAB (sec)	Execution time in FPGA (sec)	Speedup
Sequential Execution time with drawing figures	0.62254 sec (Sequential)	76696H Clock cycles (100MHz) = 0.00485014 sec	128.3550
Parallel Execution Time with drawing figures	0.62254 sec (Sequential)	70696H Clock cycles (100MHz) = 0.00460438 sec	135.2060
Sequential Execution time without drawing figures	0.017462 sec (Sequential)	C000H Clock cycles (100MHz) = 0.00049152	35.5265
Parallel Execution Time without drawing figures	0.017462 sec (Sequential)	6000H Clock cycles (100MHz) = 0.00024576	71.0530

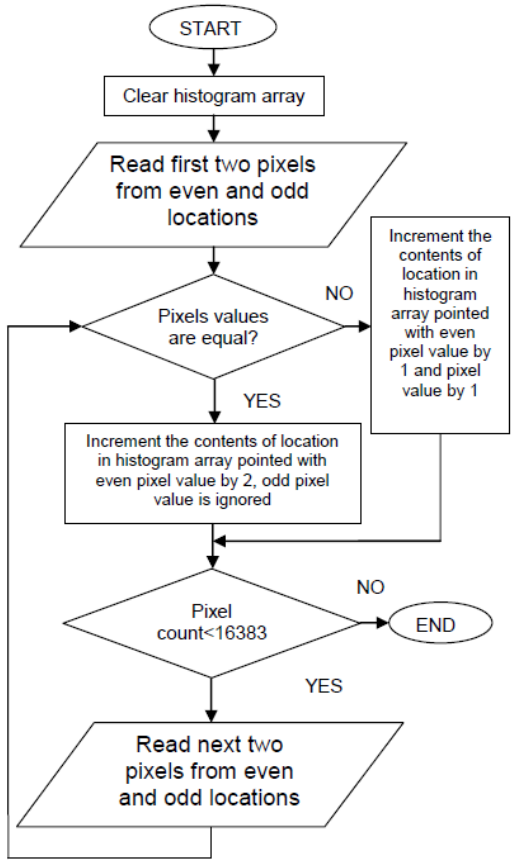


Fig. 4: Parallel histogram computation

5. Results

The histogram calculation (serial and parallel) was implemented using MATLAB, serial in and parallel in FPGA. Table 1 illustrates the execution time using MATLAB, Serial and sequential using FPGA with and without drawing processes with speedup. It is cleared that the execution time using parallel method using FPGA is faster than sequential method and the execution time using serial method using FPGA is faster than MATLAB.

The speedup is (128.3550) between MATLAB and sequential on FPGA, while speedup is (135.2060) between MATLAB and sequential on FPGA (with drawing image and histogram).

The speedup is (35.5265) between MATLAB and sequential on FPGA, while speedup is (71.0530) between MATLAB and sequential on FPGA (only histogram computation).

Table 2 shows that the speedup between serial and parallel methods on FPGA is 2. It means that the speedup was ideal.

Table 2: Speedup between serial and parallel methods on FPGA

	Sequential Execution time using FPGA (sec)	Parallel Execution time using FPGA (sec)	Speedup
Only histogram computation	C000H Clock cycles (100MHz) = 0.00049152	6000H Clock cycles (100MHz) = 0.00024576	2

The evaluation environment and evaluate the performance of the proposed hardware technique then compare it with the performance of the MATLAB implementation. Fig. 5 clarifies the execution in MATLAB program with drawing figures (Image and Histogram), while Fig. 6 illustrates the execution in MATLAB program without drawing figures.

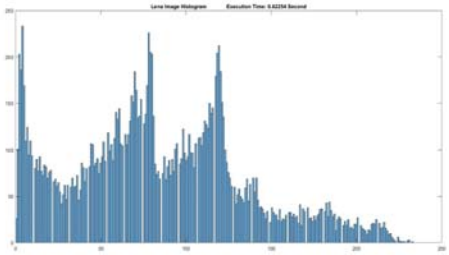


Fig. 5: Histogram implementation in MATLAB with drawing figures (Image and Histogram)

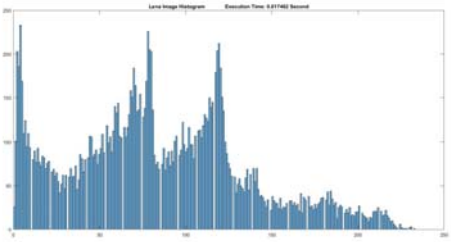
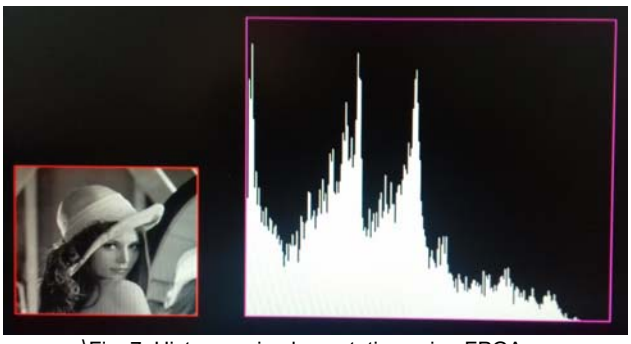


Fig. 6: Histogram implementation in MATLAB without drawing figures

Fig. 7 has been implemented using VHDL that utilized Xilinx FPGA3E embedded dual-ported block memories (BRAMs) primitives.

The BRAM width for image data corresponds to the size of the utilized bpp (bit per pixel) and the depth corresponds to the number of pixels in an image. These parameters are changed for each particular bit/pixel and image size.

As already mentioned, the total capacity of the BRAMs in the FPGA-3E is 16×1Kbytes = 16384 bytes. This capacity is not sufficient to implement large image sizes, Therefore, the memory capacity of the FPGA device is a limitation. The simplest way to overcome on this limitation is, using new devices which have much more BRAMs units. However, the number of BRAMs units in these devices is still limited.



\Fig. 7: Histogram implementation using FPGA

Fig. 8: number of clock cycles needed to implement serial histogram with drawing figures (Image and Histogram)

Fig. 9: number of clock cycles needed to implement serial histogram without drawing figures

Fig. 10: number of clock cycles needed to implement parallel histogram with drawing figures (Image and Histogram)

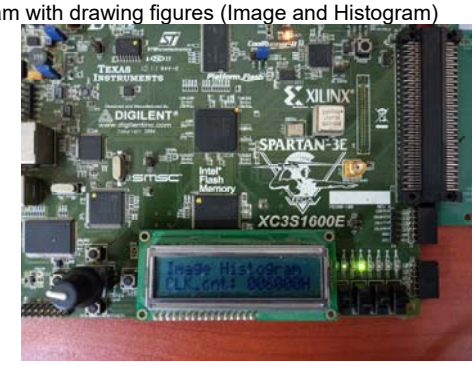


Fig. 11: number of clock cycles needed to implement parallel histogram without drawing figures

Fig.8 shows the number of clock cycles needed to implement serial histogram with drawing figures (Image and Histogram), while Fig. 9 illustrates the number of clock cycles needed to implement serial histogram with drawing figures.

Fig.10 shows the number of clock cycles needed to implement parallel histogram with drawing figures (Image and Histogram), while Fig. 11 illustrates the number of clock cycles needed to implement serial histogram with drawing figures.

Table 3 represents the chip area occupation for the Histogram designed architecture. Which is also created during synthesizes stage for Spartan-3E XC3S1600E that are used to implement the hardware unit of this design, while Spartan-3E XC3S500E recourses are not enough to implement this architecture.

Table 3: Chip area occupation for Parallel Histogram

Number of DCMs	1	8	1%
Number of BUFGMUXs	3	24	1%
Number of Dedicated IOBs	31	391	7%
Number of Block RAMs	58	520	11%
Number of Slice LUTs	942	5888	3%
Number of Slice Flip Flops	445	5888	1%
Number of Slices	488	2944	3%

6. Conclusions:

In this paper, a hardware technique for parallel histogram computation has been proposed. The novel hardware unit avoids memory collisions in the histogram functions by using a dual-ported memory. The proposed hardware calculates the histogram of an image in two phases. First, image pixels are store into block RAM each port is read address even- and odd-numbered addresses respectively. The pixel values stored in one array of matrix each pixel in image is considered indices of these histogram arrays. Therefore, in half of a cycle, these pixel values are read from dual-ported memory and in other half of the cycle updated values of histogram elements are stored. Experimental results obtained by Mat lab and FPGA implementation have shown that the proposed techniques improve the performance.

***Authors**: Riyadh Zaghlool Mahmood, University of Mosul, College of computer science and mathmatics, Software Department. Mosul, Iraq, E-mail: riyadh.zaghlool@uomosul.edu.iq*
Hiba-Aallah Tariq Abdullah, Northrern Technical University, Engineering Technical College of Mosul. Mosul, Iraq, E-mail: hibatallahtariq@ntu.edu.iq

REFERENCES

1. Asadollah Shahbahrami, Jae Young Hur , Ben Juurlink , and Stephan Wong, “FPGA Implementation of Parallel Histogram Computation”, 2nd HiPEAC Workshop on Reconfigurable Computing.
2. Murad Qasaimeh, Joseph Zambreno and Phillip H. Jones, “A Runtime Configurable Hardware Architecture for Computing Histogram-based Feature Descriptors”, 2018 International Conference on FieldProgrammable Logic and Applications.
3. Mohammad HOSSEINABADY 1 , Jose Luis NUNEZ-YANE, “Pipelined Streaming Computation of Histogram in FPGA OpenCL”, In Parallel Computing is Everywhere (pp. 632-641). (Advances in Parallel Computing; Vol. 32). IOS Press. https://doi.org/10.3233/978-1-61499-843-3-632.
4. Luca Maggiani, Claudio Salvadori, Matteo Petracca, Paolo Pagano, Roberto Saletti, “Reconfigurable architecture for computing histograms in real-time tailored to FPGA-based smart camera”, 2014 IEEE 23rd International Symposium on, Jun 2014, Istanbul, Turkey. ff10.1109/ISIE.2014.6864756ff. ffhal-01205924f.
5. Orlando J. Hernandez, Member, IEEE, “A High-Performance VLSI Architecture for the Histogram Peak-Climbing Data Clustering Algorithm”, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 14, NO. 2, FEBRUARY 2006.
6. Burak Unal, Ali Akoglu, “Resource Efficient Real-Time Processing of Contrast Limited Adaptive Histogram Equalization”, 2016 26th International Conference on Field Programmable Logic and Applications (FPL).
7. Ernest Jamro, Maciej Wielgosz, Kazimierz Wiatr, “FPGA implementation of strongly parallel histogram Histogram Equalization” 1-4244-1161-0/07/$25.00 ©2007, IEEE
8. Shahbahrami, Ben Juurlink, Stamatis Vassiliadis, “SIMD Vectorization of Histogram Functions”, 2007 IEEE International Conf. on Application-specific Systems, Architectures and Processors (ASAP).
9. Krishna Swaroop and Gautam Uurmi, Solutions Pvt. Ltd., “Parallel Histogram Calculation for FPGA”, 2016 IEEE 6th International Conference on Advanced Computing (IACC).
10. H.D. Cheng and X.J. Shi, “A simple and effective histogram equalization approach to image enhancement”, Digital Signal Processing 14 (2004) 158–170
11. Komal Vij, Yaduvir Singh, “Enhancement of Images Using Histogram Processing Techniques”, Komal Vij,Yaduvir Singh Int. J. Comp. Tech. Appl., Vol 2 (2), 309-313
12. Ernest Jamro, Maciej Wielgosz, Kazimierz Wiatr, “FPGA Implementation of Strongly Parallel Histogram Equalization”, 1-4244-1027-4/07/$25.00 ©2007 IEEE