FPGA Implementation of Strongly Parallel Histogram Equalization

Ernest Jamro, Maciej Wielgosz, Kazimierz Wiatr
AGH University of Science and Technology
al. Mickiewicz 30, 30-051, Poland
Academic Computer Center CYFRONET
ul. Nawojki 11, Kraków 30-950, Poland
email: jamro / wielgosz / wiatr @agh.edu.pl

Abstract - **Highly parallel architecture for local histogram equalisation is studed. Three different kinds of approaches to the parallel architecture are regarded in this paper. 1) Module-level – which focuses on processing as many data as possible within a single module. 2) 1D – Several modules conducting simultaneously histogram equalization on partially overlapping (either horizontally or vertically) frames. 3) 2D – utilizes the same approach as 1D but in two dimensions. Beside above-mentioned solutions, differencial processing of overlapping frames was also considered. At the end of this paper the optimal proportion of the above mention solutions are studied and implementation results given.**

I. Introduction

Histogram equalization is a relatively simply operation and is often performed by general-purpose processors. Nevertheless it may require hardware acceleration to satisfy the high speed requirements for real-time systems. Face detection systems [1] based on neural networks may serve as an example. The neural network is fed with square-shape 20×20 pixel frames of an input picture. The frame (mask) is gradually moved by 1 pixel step by step over the whole input image in order to detect whether it contains any faces. In the case of 512×512 pixel input image, this operation is conducted 492×492 times on 20×20 pixel frames. Local histogram equalization operation must be performed before the frame is passed to the neutral network. Histogram calculation is one of processes conducted during histogram equalisation [2]. Consequently, histogram calculation for a single 512×512 picture requires $492\times492\times20\times20\approx 10^8$ counts of input pixels. It is worth to mention that the above evaluated number refers only to original scale of the picture. The same calculations should be done for the sequence of gradually decreased scale picture to evaluate the total number of pixel counts. Summing up, hardware solution combined with strong parallel architecture is designed to speed up the calculation of histogram. Unfortunately, the operation of histogram calculation in contrast to neural network [3] cannot be easily parallelized because of sequential counting process of input pixels. Therefore this paper introduces several solution of parallel architectures adapted to histogram equalization module.

The hardware implementation of parallel histogram equalisation, up to the authors' knowledge, has not been considered in literature. The examples of histogram calculation implementations can be found in [4, 5]. Calculation time required by these systems is too large for our case, therefore some enhancements have to be introduced.

In-module parallel histogram calculation will be described in the first part of this paper. For this architecture a single module can accept several input pixels in a single clock cycle. In Section IV studies differential histogram calculation. The result of histogram calculation for neighboring frames is partially the same as the input frames overlap. Consequently differential histogram calculation utilizes the above fact and therefore calculations are conducted only for not-overlapping input pixels. Section V considers multi-module parallel architecture for which several parallel modules operates on the same (partially overlapping) input data. Two different options: 1-dimentional and 2-dimentional are further studied. Section VI describes interference between differential histogram calculation and multi-module parallelism. Section VII considers cooperation between histogram calculation and LUT conversion. The last chapter of this paper includes implementation results of these system.

II. Hardware Implementation of Histogram Calculation

At the first glance the histogram calculation module is regarded as composed of Block RAM (BRAM) memory and incremental logic as it is shown on Fig. 1A. Module operates as follows: Input data (of which histogram is being evaluated) address the BRAM. The BRAM *data_out* shows the count of *DataIn's* prior occurrences at the BRAM address bus. Then this count is incremented by one and written back to the BRAM at the same address; and so on.

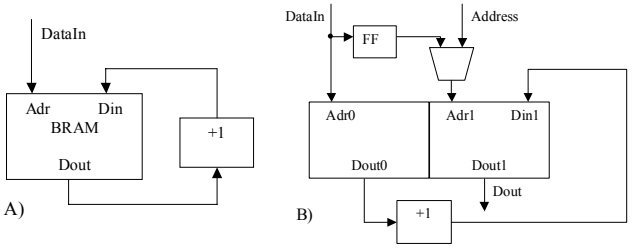

Fig. 1. Opb_hist block diagram; A) Simplified diagram, B) real diagram

Unfortunately, the BRAM memory limits the calculation speed as the BRAM output data is one clock cycle delayed in comparison to the address bus (a synchronous memory data

This work was support by Polish scientific research funds

read). Consequently, evaluation of a single input pixel involves two clock cycles. A similar system was proposed in article [5].

In order to speed up the above circuit to evaluate a single input pixel in a single clock cycle, a dual ports BRAM in employed [4] (Fig. 1B). The first port is utilized to read the prior input data count, the second one is dedicated to write back the incremented count. An additional multiplexer is employed (addressed with *Adr1* signal) to enables reading of calculated histogram.

III. In-module Parallelism

In order to speed-up the histogram calculation process, in-module parallelism is introduced, which operates on many input pixels at once. An example of two parallel units system is presented in Fig 2. Two input data (*DataIn0, DataIn1*) are fetched at a single clock cycle. The resultant histogram is an aggregate of two fractional operation results. The disadvantage of this solution is that the histogram calculation time is comparable to histogram reading time. The calculation time is 20×20/2= 200 clock cycles for 20×20 pixel input frame and two parallel units. Reading the result and setting to zero all memory cells take 2×256 clock cycles for 8-bit pixels. Therefore the more histogram calculation sub-modules work together the worse proportion between calculation and read time is obtained. Read time becomes the dominant part. Therefore parallel connecting of many histogram calculation units is not advisable.

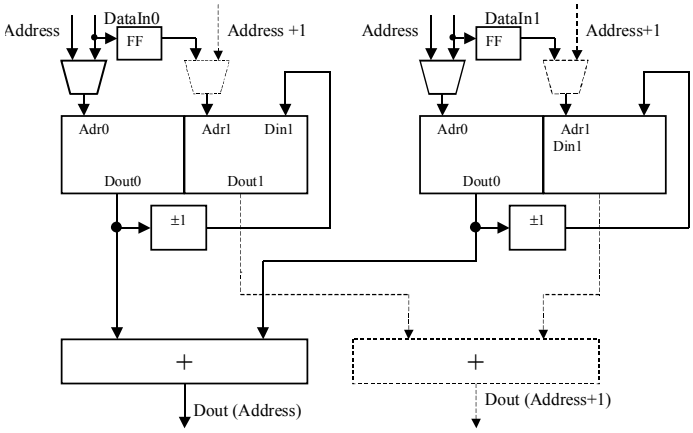

Fig. 2. Block diagram of parallel histogram calculation (solid line) and reading (dashed line)

Fortunately, reading process can also be spped-up. The first solution, shown in Fig. 2, is based on utilization of two concurrent BRAM ports and therefore it doesn't involve much more additional hardware. Two additional multiplexers are essential, one of them at *Adr0* port and the second one at *Adr1*. As a result of this modification the read time is reduced by a half. Unfortunately, the number of BRAM ports is limited to two, therefore further reading seed-up is not available for this method.

Consequently to further speed-up the reading process, the number of BRAM memory should be increased. In this method, the whole memory is regarded twofold by the rest of the system. During histogram calculation process BRAM memory is considered as one large memory whereas during the reading process the same memory is regarded as composed of several independent smaller BRAMs. In case of two BRAM's, odd pixels (odd numbered pixels) are read from the first memory while even pixels are read from the second memory. Instance of such system is presented in Fig. 3.

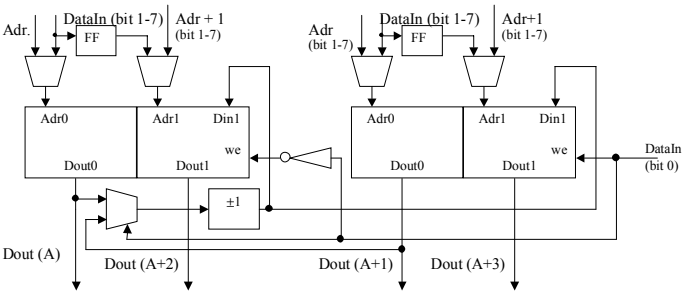

Fig. 3. Parallel histogram reading (4 reads at the time).

Ability to use smaller memories can be considered as certain benefit from parallel reading. Unfortunately, the size of BRAM memory available in FPGA is so large that decreasing the size of a single memory does not result in fall down of the total number of used BRAMs. Alternative solution is utilization of distributed memories 16×1 included in CLB (Configurable Logic Block). Consequently for highly parallel reading (factor of about 8 or more) the size of a single memory used for reading is so small (32-bit deep or less) that the distributed RAM should be employed.

For Xilinx Virtex2, BRAM memory size is 18kb, and the data bus width is up to 36-bit. Consequently instead of using 4 independent 256×9-bit BRAMs a single 256×36-bit BRAM can be used for reading. This significantly reduces the total number of BRAMs. Consequently a single 256×36-bit dual port BRAM can be used for up to 8-parallel reads.

It is worth to emphasize that parallel approach to reading and calculation processes are independent of each other. Consequently, the total number of utilized BRAMs is a product of the number of BRAMs employed in parallel histogram calculation and parallel histogram reading. Therefore it is not profitable to apply both calculation and read parallelism.

In addition to the above drawback, highly parallel histogram reading makes difficult cooperation between other modules. This holds, for instance, for Look-Up Tables (LUT) which are programmed according to the data read from the histogram module. Parallel writing the LUT memory suffers from the similar phenomena as it is the case for the parallel histogram reading.

IV. Differential Histogram Calculation for Neighboring Frames

Face detection system considered in this paper requires histogram equalization for neighboring frames. Consequently, histogram that has been calculated in the previous step (for the preceding frame) can be utilized to evaluate histogram for the next frame (one pixel right). Hardware module should be able

to increment and decrement the number of pixels to perform differential histogram calculation. This operation is presented in the Fig. 4.

An input frame can be divided into three part:

A) Decrement area – this part of the picture belonged to the previous frame but does not belong to the presently evaluated frame. Pixels which are lying in this area, are counted down during histogram calculation.

B) Area that is directly copied – the common part of the previously and presently calculated histogram. No operations are conducted on this area.

C) Incremental area – area that does not belong to previous frame but it is a part of presently calculated frame. Standard histogram calculation is conducted for this area (increment).

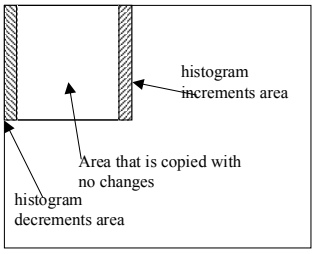

Fig . 4. Different frame areas utilized by differential histogram calculation

It can be easily noticed from Fig. 4, that differential histogram calculation requires significantly fewer operations. For $W \times W$ pixels frame, only $2 \times W$ input pixels are required to calculate the histogram of the new frame. This holds as the differential histogram is decremented by W pixels (area A) and incremented by W pixels (area C).

An additional advantage of differential calculation is no need to reset BRAM memory cells before the new frame starts to be calculated. Therefore the histogram reading time is also shorten by a half (the reset process is included to reading time). Nevertheless the read time is still significantly longer than the calculation time. Therefore the recommended parallel ratio for histogram calculation is 1 (a single input pixel at the time). For histogram reading, the parallel ratio is 2 (or 4) for BRAM 256×16 (Virtex) or up to 8 for 256×36 BRAM (Virtex 2 or latter).

V. Multi-module Parallelism.

In-module high-level parallelism is not very efficient because the hardware requirements rapidly grows with the growing parallelism level. Besides, the external memory interface often limits the maximum number of input pixels fed to the histogram. Therefore multi-module parallelism is regarded as a more convenient method to speed-up the calculations. For multi-module parallelism, histograms of many neighboring frames are calculated at a time. It is recommended to calculate neighboring frames because input pixels of neighboring frames partially overlap each other, therefore single external memory interface is employed. One dimension and two dimension parallel approach will be presented in the forthcoming sections.

A. One Dimension Parallelism

Computations are conducted concurrently for either vertical or horizontal neighboring frames. Let us evaluate the number of input pixels required to calculate histograms for N frames. It can be easily seen that for $W \times W$ pixel input frame, the total number of pixels P_{1D} that must be fetched is as follows:

$$P_{1D} = W \cdot (W+N-1). \tag{1}$$

$$P'_{1D} = W \cdot \left\lceil \frac{W+N-1}{N} \right\rceil \cdot N \tag{1a}$$

Eq. (1a) considers the padding effect when the first and (or) the last word in the line are not fully utilize, i.e. N- input pixels are fed at the time and usually it is not possible to break the word alignments.

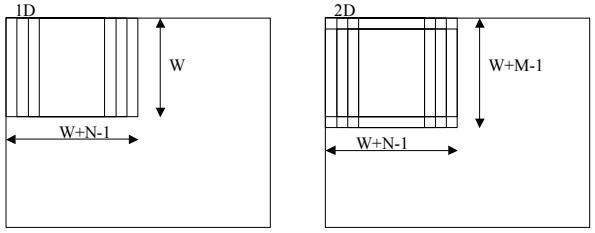

Fig . 5. Input data for histogram calculation,1D – horizontally, 2D – vertically and horizontally.

Unfortunately, all parallel modules cannot work concurrently all the time (not all pixels are valid for every module), therefore average utilization of each module is given as follows:

$$R_{1D} = \frac{W \cdot W}{P_{1D}} = \frac{W}{W+N-1} \tag{2}$$

$$R'_{1D} = \frac{W \cdot W}{P'_{1D}} = \frac{W}{\left\lceil \frac{W+N-1}{N} \right\rceil \cdot N} \tag{2a}$$

It can be easily noticed from (2) that the best utilization of parallel modules is obtained for small N. For small N ($N << W$) average utilization is close to one and parallelism efficiency is high. Unfortunately, the calculation speed $N \cdot R$ saturates relatively fast and this method is not efficient for $N > W$. It is worth to mention that the above equation refers only to histogram calculation time, read and reset time is ignored hereby.

B. Two Dimensions Parallelism

For 2-D parallelism computations are conducted concurrently for both vertical and horizontal neighboring frames. In general case N vertical and M horizontal parallelism can be employed. Consequently, the total number of parallel modules is $N \times M$. The number of input pixels fetched for this solution is given as follows:

$$P_{2D} = (W+N-1) \cdot (W+M-1). \tag{3}$$

$$P'_{2D} = \left\lceil \frac{W+N-1}{N} \right\rceil \cdot N \cdot \left\lceil \frac{W+M-1}{M} \right\rceil \cdot M \tag{3a}$$

Eq. 3a (similarly like eq. 1a) considers the padding effect. Average utilization R_{2D} of each module is given as follows:

$$R_{2D} = \frac{W^2}{P_{2D}} = \frac{W^2}{(W+N-1)\cdot(W+M-1)} \quad (4)$$

$$R'_{2D} = \frac{W \cdot W}{P'_{2D}} = \frac{W \cdot W}{\left\lceil \frac{W+N-1}{N} \right\rceil \cdot N \cdot \left\lceil \frac{W+M-1}{M} \right\rceil \cdot M} \quad (4a)$$

Comparison of average utilization R'_{1D} and R'_{2D} for different parallel approaches: 1D and 2D and the same number of computational units is presented in Fig 6.

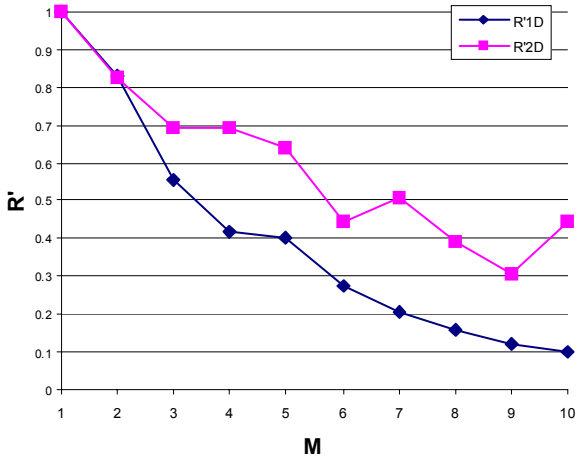

Fig. 6. Average utilization R in the function of the number parallel units 1D) N = M^2; 2D) N = M

According to Fig. 6 for 1D and $M = 4$ ($N = M^2 = 16$ parallel units) average utilization $R_{1D} = 0.42$. Conversely, for 2D approach $N=M=4$ (the same number of parallel units as for 1D) average utilization $R_{2D} = 0.7$. Consequently for the larger number of units, the 2D approach is more efficient.

VI. Parallelism vs. Differential Histogram Calculation

Parallelism impacts strongly deferential histogram calculation. It is recommended to implement differential calculation vertically and parallel calculation horizontally. This method will be further denoted as 1DD. To calculate 1DD histogram it is essential to deliver P_{1DD} input pixels:

$$P_{1DD} = 2\cdot(W+N-1) \quad (5)$$

$$P'_{1DD} = 2 \cdot \left\lceil \frac{W+N-1}{N} \right\rceil \cdot N \quad (5a)$$

It is worth to note that the combination of parallel 1D and differential calculation does not disturb each other. This is very important feature, which stays quite opposite to 2D parallel approach and differential histogram calculation, denoted further as 2DD. In the case of 2DD approach disturbances cause significant decrease in effectiveness. It has been illustrated in Fig. 7.

The number of input pixels which must be delivered during histogram calculation (sum of area A and C in Fig. 7) is increased and is given as follows:

$$P_{2DD} = (W+N-1)\cdot 2\cdot M \quad (6)$$

$$P'_{2DD} = \left\lceil \frac{W+N-1}{N} \right\rceil \cdot N \cdot 2 \cdot M \quad (6a)$$

While analyzing (6), it is easy to discern that for a constant number of parallel units ($N \cdot M = const$) optimal solution is reached by increasing N and remaining $M = 1$, which is indeed implementation of 1DD.

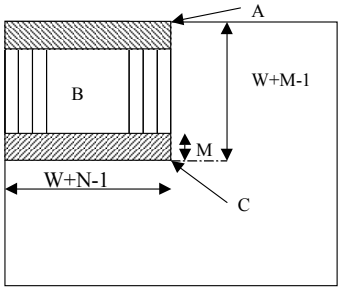

Fig. 7. 2DD parallel approach

VII. Histogram Calculation and LUT Conversion

A. In-Module Parallelism

Parallel LUT conversion experience similar parallelism problems as histogram calculation. Parallel writing (programming) new LUT contents can be achieved in similar way as parallel histogram reading, the main difference is the data bus width which is 8-bit. Consequently using a dual port 256×16-bit BRAM (Xilinx Virtex), four LUT address locations can be written in a single clock cycle – a single 16-bit data bus performs two separate 8-bit writes in a single clock cycle. For a 256×32-bit dual port BRAM (Virtex2 or latter) up to eight LUT writes can be achieved in a single clock cycle. Increasing a number of parallel writes requires several BRAMs to be used during LUT programming and these BRAMs are then merged into a single large BRAM during LUT conversion operation. Similar architecture was used for strongly parallel histogram reading and resetting.

Parallel LUT conversion is conducted in similar way as histogram calculation. Nevertheless, a dual port BRAM can conduct up to 2 parallel LUT conversions, therefore the number of BRAMs is halved. Greater level of parallelism requires several BRAM modules to be used in similar way as it was the case for histogram calculation. Unfortunately, the differential LUT conversion is not possible as it is the case for histogram calculation. It should be noted that the total number of BRAMs is the product of the number of BRAMs required for parallel LUT programming and LUT conversion operations. Summing up, high-level in-module parallelism for both LUT programming and conversion is not recommended.

Histogram equalization can be divided into three different stages:

1) histogram calculation
2) reading previously calculated histogram and programming LUT memory, in this step just read histogram data are used to calculate a new LUT conversion table.
3) LUT conversion

B. Multi-Module Parallelism

As it has been described in the previous sections in-module parallelism quickly saturates – requires much greater hardware resources for relatively small speed-ups. Consequently multi-module parallelism is further studied. Parallel multi-module histogram equalization behaves in similar way as it is the case for multi-module histogram operation. The main difference is that differential LUT operation is not feasibly. Optimal solution for LUT conversion according to Fig. 6 and eq. 4 is 2D parallel approach ($M=N$). Conversely for differential histogram calculation the optimum solution is for $M=1$ (see Section VI).

To make cooperation between histogram and LUT module effective, the number of parallel units N and M should be the same for both histogram and LUT operation. Unfortunately optimal performance condition for histogram and LUT operations are different and both operations have to be taken under consideration in order to implement an effective system. Let us make an assumption that optimal solution is obtained for the circuit for which the minimum input data transfers P is obtained. The total number of fetched input pixels P is the sum of input pixels P_{2DD} (see (6)) required by differential histogram calculation and the number of input pixels required by 2D LUT conversion P_{2D} (see (3)):

$$P = P_{2DD} + P_{2D} = (W+N-1)\cdot(W+3\cdot M-1) \qquad (7)$$

From (7) for the constant number of parallel units $L = N\cdot M$, the minimum number of input transfers is obtained for:

$$N = \sqrt{3\cdot L} \qquad M = \sqrt{L/3} \qquad N = 3\cdot M. \qquad (8)$$

It should be noted that (8) holds for systems for which external memory interface limits the whole system throughput. Otherwise, a designer has to find out whether increase in-module or multi-module level of parallelism. In this case AT product (product of occupied area and calculation time) should be considered.

Tab. 1 presents recommended solutions for different calculation times t (average number of clock cycles required to compute a 20×20 frame). Column 2 ($MAX(t_{HD}, t_L)$) presents the number of clock cycles required for simultaneous differential histogram calculation t_{HD} and LUT conversion t_L. Column 3 (t_P) presents the number of clock cycles required to read histogram and program LUT memory. Column B_H and B_L denotes the number of BRAMs required for a single module to calculate histogram and LUT respectively. B_{tot} denotes the total number BRAMs (including all modules). Columns N, M – presents horizontal and vertical multi-module parallelism. Column AT denotes AT product.

From Tab. 1 the following conclusions can be drawn: decreasing the calculation time causes that at first the in-module parallelism is increased up to the level when up to the 8 LUT writes or 4 (or even 8) LUT conversions are carried out in a single clock cycles. Then the multi-module parallelism is introduced at first 1D (M=1) and then 2D. Introducing 2D parallelism (M>1) causes that differential histogram calculation requires more hardware resources, thus M is significantly smaller than N. Increasing multi-module parallelism causes that the efficiency drops (AT product increase) consequently also in-module parallelism also should increase. Summing up, the following conclusion can be drawn: decreasing calculation time below about $t<12.25$ causes significant increase of the AT product.

TABLE 1.
RECOMMENDED IN-MODULE AND MULTI-MODULE PARALLELISM FOR DIFFERENT CALCULATION TIMES

t	MAX (t_{HD}, t_L)	*tp*	B_H	B_L	B_{tot}	N	M	AT
656	400	256	1	1	2	1	1	1312
456	200	256	1	1	2	1	1	912
328	200	128	1	1	2	1	1	656
264	200	64	1	1	2	1	1	528
232	200	32	1	1	2	1	1	464
164	100	64	1	2	3	1	1	492
132	100	32	1	2	3	1	1	396
82	50	32	1	4	5	1	1	410
71	55	16	1	2	6	2	1	426
72	40	32	1	8	9	1	1	648
57	25	32	2	8	10	1	1	570
43.5	27.5	16	1	4	10	2	1	435
38	30	8	1	2	12	4	1	456
23	15	8	1	4	20	4	1	460
20.5	16.5	4	1	2	24	4	2	492
14	10	4	1	4	40	8	1	560
12.25	8.25	4	2	4	48	4	2	588
11	9	2	2	2	64	4	4	704
7.5	5.5	2	2	4	96	8	2	720
6.5	4.5	2	4	4	128	4	4	832
4.75	2.75	2	4	8	192	8	2	912
4	3	1	4	4	256	8	4	1024
3.5	2.5	1	2	8	320	32	1	1120
3.25	2.75	0.5	2	4	384	32	2	1248
2.75	2.25	0.5	4	4	512	16	4	1408
1.875	1.375	0.5	4	8	768	32	2	1440
1.625	1.125	0.5	8	8	1024	16	4	1664
1	0.75	0.25	8	8	2048	32	4	2048

VIII. IMPLEMENTATION OF HISTOGRAM EQUALIZATION

New LUT programming values are calculated according to equation:

$$v_k = \frac{K-1}{W^2}\sum_{i=0}^{K-1} Histogram[i] \qquad (9)$$

where: K- levels of grayscale (K=256 for 8-bit image), W-frame size.

The module conducting base operation of histogram equalization is shown in Fig. 8. The system has been implemented in Xess evaluation Board [6].

The main module denoted *Hist_equ_block* is comprised of three parts: Histogram calculation module, Equation module and the LUT memory (two port BRAM memory). Through *DataIn0* input data is being fed to histogram module. After histogram has been calculated both histogram module outputs

are utilized to read data (Dout0 and Dout1) and send them do *equalization* module. After equation operation has been performed, data is passed to the BRAM memory. There are two multiplexers connected to the BRAM memory, which enable BRAM memory to act twofold. Firstly, BRAM memory is addressed by *equalization* module to program BRAM memory. Secondly, to perform LUT operation, i.e. after the BRAM has been properly programmed in the first step it is addressed by the input frame pixels.

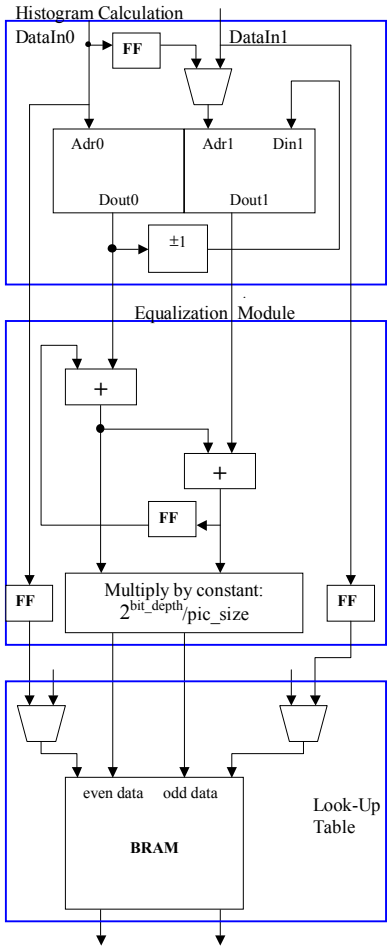

Fig. 8. *Hist_equ_block* – block diagram of histogram equalization logic

Implementation results of *opb_hist_equ* are presented in Tab. 2. Column 2 presents the total area occupied by histogram equalization module with only a single *hist_equ_block*. Column 3 presents the area of the *hist_equ_block* alone (see Fig. 8). It is easy to estimate the resources absorbed when more than one *hist_equ_block* modules are implemented since the difference between column 2 and 3 is consumed by control logic, which is relatively constant for different number of *hist_equ_block* modules.

In this paper area consumption is approximated by the number of BRAMs. This assumption is made as the total number of occupied LUTs (CLBs) comparison to the available FPGA LUT resources is very low. This does not holds for BRAM memories.

The design module is compatible with IBM On-chip Peripheral Bus (OPB) and Xilinx Embedded Development Kit (EDK). The OPB and EDK are employed for easy connection of on-chip peripheral devices. They provides a common design point for various on-chip peripherals. Modular design methodology can be easily adopted and the designed system can be easily extended or modified.

TABLE 2.
IMPLEMENTATION RESULTS

Area	Whole circuit	*hist_equ_block*
4-input LUT	96	62
Slices	64	41
BRAM	2	2

IX. CONCLUSIONS

In this paper the following aspects of parallel architectures are considered:

- In-module parallelism, when a single module can accept several input data in a single clock cycle. The main drawback of this architecture is that the total number of BRAMs is the product of the number of BRAMs required for parallel histogram calculation (or LUT programming) and histogram reading (LUT conversion). Consequently, the level of in-module parallelism is rather limited.
- Multi-module parallelism, for which the same input data are used by different neighboring modules. In this case 1-Dimenctional (1D) and 2D parallelism should be considered. For high level of parallelism 2D is much more efficient.
- Differential histogram calculation which exploits the fact that neighboring frames partially overlap. Consequently only difference between two neighboring frames need to be considered. Unfortunately differential histogram calculation influence 2D multi-module parallelism and can be applied only for histogram (not for LUT conversion). Consequently a trade-off between multi-module and differential calculation should be considered.

Summing up, the optimal parallel architecture are presented in Tab. 1. The parallelism gives a huge speed-up to the system but one has to keep in mind to employ appropriate parallelism approach that will match throughput of the system.

REFERENCES

[1] Rowley H. A., Baluja S., Kanade T., *Neural Network-Based Face Detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 1, January 1998, pp. 23-38

[2] Gonzalez, R. , Wintz P., *Digital Image Processing,* Addision-Wesley 1987.

[3] Jamro E., Wiatr K. *A Novel Parallel-Serial Architecture for Neural Networks Implemented in FPGAs,* Proc. of IEEE Design and Diagnostics of Electronics Circuits and Systems Workshop, Sopron, 13-16 Apr. 2005, pp.121-128.

[4] Jamro E., Wielgosz M., Wiatr K., *FPGA Implementation of the Dynamic Huffman Encoder*, Proc. IFAC Workshop on Programmable Devices and Embedded Systems, Brno, Feb. 14-16, 2006, pp.60-65

[5] Garcia E. *Implementing A Histogram for Image Processing Applications,* Xcell Journal Online, Xilinx: xcell38_46.pdf.

[6] Xess Corp. *XSV Board V1.1 Manual,* 2001, www.xess.com