Reconfigurable architecture for computing histograms in real-time tailored to FPGA-based Smart Camera

Luca Maggiani[1,2], Claudio Salvadori[1], Matteo Petracca[2], Paolo Pagano[2], Roberto Saletti[3]

[1]TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy
[2]National Laboratory of Photonic Networks, CNIT, Pisa, Italy
[3]Dipartimento Ingegneria dell'Informazione, University of Pisa, Pisa, Italy

***Abstract*—The design and development of distributed innovative services leveraging pervasive smart camera network solutions requires the use of reconfigurable low-cost smart cameras. In this respect, FPGA based Smart Cameras enabled to wireless communication that follow the Internet of things paradigm are a promising solution. The paper proposes an optimized design of the histogram extractor algorithm targeted to low-complexity and low-cost FPGA based Smart Cameras. The proposed solution is the basis for a wide range of distributed computer vision applications. We first define a general architecture for the image histogram core, then we evaluate its performance with a real implementation.**

I. Introduction

Smart Camera Network (SCN) is the natural evolution of state-of-the-art centralized computer vision applications towards distributed and pervasive systems. Differently from Wireless Sensor Networks (WSNs) which generally perform basic sensing tasks, SCNs consist of autonomous devices, Smart Cameras (SCs), performing on-board image processing algorithms strongly optimized to fit to the limited amount of available resources [1].

One of the biggest efforts in designing pervasive SCNs based on low-end devices is the porting of complex, and computational intensive, computer vision algorithms to resource constrained embedded devices. For instance, an optimized/approximated version of Gaussian Mixture Model (GMM) is proposed in [2], with the goal of a real instantiation in low-complexity micro-controllers lacking Floating Point Unit (FPU). Although the authors proved that a micro-controller can reach real-time performance in performing GMM-based tasks, the limits of such CPU-based solution in performing heavy computer vision tasks have been outlined. An innovative architecture for smart cameras acting as nodes of a network following the *Internet of Things* (IoT) paradigm has been proposed in [3], to overcome the computational limits in the SCs design. In such a work, a visual IoT node (i.e., IoT smart camera) consists of a micro-controller and an FPGA. The former is in charge of managing the network communications and the high-level organization of the computer vision pipeline. The latter handles the heavy processing operations (i.e., pixel-wise or machine learning algorithms) according to the streaming paradigm (data are processed directly when they appear at the input port without any buffering stages). A key feature of the proposed SC architecture is a flexible and reconfigurable FPGA internal structure, managed by a SoftCore through the CPU data bus.

A challenging computer-vision algorithm to be ported to FPGA-based smart cameras is the *Histogram of Oriented Gradient* (HOG) [4]. The algorithm basically extracts image features by deploying a histogram of the pixel edge directions, with the aim of identifying pedestrians in the scene. Even though the histogram software implementation is a simple task used in several light-weight detection algorithms [5], [6], the hardware realization of such algorithm in not a trivial task. A possible hardware oriented histogram extractor has been proposed in [7], [8]. The data dependency issues are directly managed at the hardware level in these cases, by deploying a linear pipeline elaboration cell. On one side this approach has the advantage of processing multiple data in a single clock cycle, on the other side it represents a dedicated solution designed to process buffered data.

This paper proposes an optimized design of the histogram extractor algorithm targeted to low-complexity and low-cost SCs node based on a mid-range FPGA. This histogram core can work with a continuous input data flow, thus following the streaming paradigm. As pointed out in [9], parallel histogram computation requires a complete algorithm redesign. In this context, our work removes the memory access conflicts from the histogram computation, leading to more flexible and efficient ways to exploit parallelism. This solution reaches better performance with respect to the state-of-the-art solutions, while being fully compatible with the reconfigurable SC architecture proposed in [3]. More in detail, we first define a general architecture for the image histogram core in this paper, then we evaluate its performance in terms of FPGA resource occupancy, memory footprint and latency.

The rest of the paper is organized as follows. A detailed description of the methodology followed to design the histogram core is given in Sec. II. The implementation details and the performance results are shown in Sec. III. Conclusions are finally highlighted in Sec. IV.

II. Methodology

As briefly introduced in Sec. I, the realization of a histogram core is a challenging task, and requires a complete algorithm re-design with respect to a software-oriented solution. Indeed, in low-complexity FPGA-based SCs performing

real-time image processing a high degree of parallelism is necessary to process data directly when they appear at the input port, in order to minimize the output latency. Moreover, the proposed histogram core has to be both reconfigurable and integrated with other blocks, to satisfy the requirements of an IoT oriented design compliant with the architecture shown in [3]. Following these main requirements, the proposed histogram core can be seen as a black-box with two inputs, the data-stream and the data-valid signals (to trigger a new data acquisition), and one output, the histogram stream to be stored into the memory. Moreover, as proposed in [3], the block is connected to the CPU data bus, thus enabling a direct communication with the SoftCore in charge of managing the reconfiguration tasks.

In general, a simple histogram computation core can be represented according to the diagram depicted in Fig. 1. In this configuration each pixel updates the value of the corresponding histogram bin by performing a read-modify-write operation. The whole operation requires four clock cycles per pixel. Thus, if the first pixel appears as input at time T_0, the complete histogram is available as output at the instant $T_0+4\cdot N_p\cdot T_p = T_0+4\cdot T_f$, where N_p is the number of pixels of the whole image, T_p is the pixel period, and T_f the frame period. The above described histogram evaluation method is not based on a streaming paradigm, since the processing of a new pixel requires the completion of the previous pixel, and strongly limits the development of real-time image processing applications.

We propose a new circuit based on the module depicted in Fig. 2 and called *histogram-subcell*, to reduce the computational time for evaluating the image histogram. This block is essentially composed by a dual-port RAM memory array, which has the same size of the histogram, and a Finite State Machine (FSM), which controls the data flow. By deploying a dual-port RAM module, the system can jointly execute read and write operations in the same clock cycle. At the same time, the internal FSM (subcell FSM in Fig. 2) controls the elaboration as a sequence of pipeline operations (reported in Fig. 3): *READ*, *LOAD*, *ADD*, and *WRITE*. When a pixel comes to the input port, a data valid signal is enabled and the internal FSM starts a new elaboration cycle. In the READ state the circuit reads the bin addressed by the current pixel value and then stores the bin content in a register during the LOAD state.

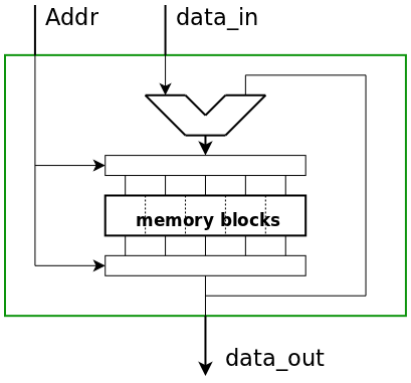

Figure 1: Typical histogram computation block.

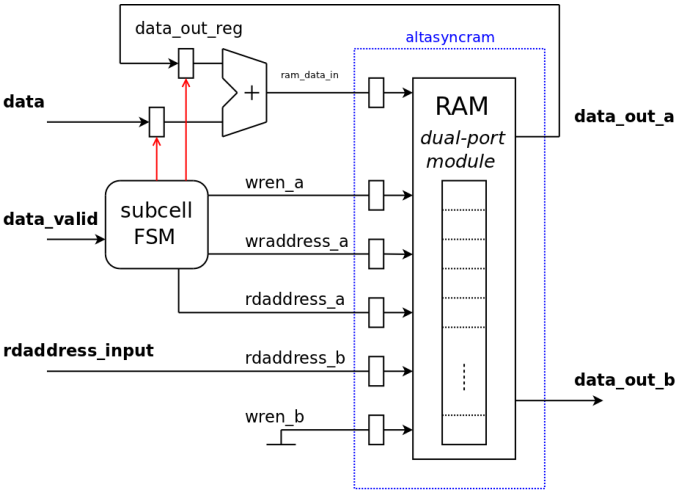

Figure 2: Histogram-subcell module.

In the ADD state, the bin value is updated by an incremental operation and finally the memory array is updated with the new value during the WRITE state. The transitions occur every clock cycle.

Though the designed pipeline architecture supports the streaming paradigm (every FSM state needs only one clock cycle to be executed), it introduces a memory access conflict to be solved. Indeed, the above described FSM does not provide an atomic read-modify-write operation. As matter of example, suppose that the i-th bin is addressed at time T_0, and that the same bin is required at the following data-valid signal, at time T_1 $(= T_0 + 1)$. The memory content in T_1 is not consistent because the previous elaboration phase is not completed and the bin value is not updated. A FSM-controlled parallel architecture called *Histogram cell* is proposed to address the above described issue. Since the histogram-subcell circuit requires four clock cycles to complete the read-modify-write iterations, we deploy four histogram-subcell instances to handle groups of four consecutive pixels as parallel flows.

The module computes four different sub-histogram data structure for the same image, where the j-th pixel updates the $(j \mod 4)$-th histogram-subcell instance. Splitting the pixel operation in parallel processing flows prevents memory access conflicts, because each histogram-subcell instance is not considered until the previous operation has been completed (i.e., four clock cycles). The Histogram cell behavior is shown in Fig. 5.

When an image is completely acquired, the four sub-

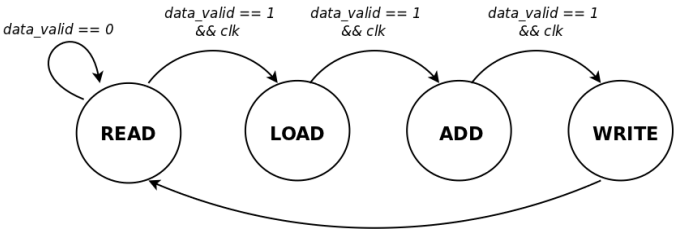

Figure 3: Histogram-subcell FSM.

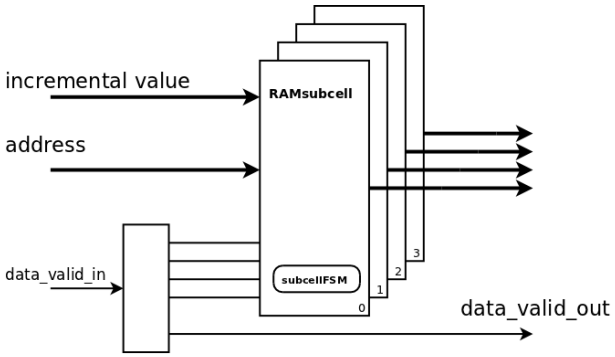

Figure 4: Histogram cell.

histograms has to be merged into the complete histogram. The final histogram can be obtained by using a final adder stage to sum the sub-histogram memory cells each other (i.e., the sub-histogram bin) sequentially: this last hardware block introduces a latency of $N + 2$ clock cycles, where N is the number of bins. The *Histogram core* block diagram is depicted in Fig. 6. By considering the latency introduced by each block of the Histogram core, the overall latency L_{total} (after the arrival of the last image pixel) can be evaluated as:

$$L_{total} = N + 6 \tag{1}$$

The total latency depends on the number of bins plus 6 clock cycles (4 for the FSM plus 2 for the final adder stage). The histogram is completed after the adder stage and is eventually stored in memory.

Advantages with respect to the state-of-the-art solutions [7], [8] are hereby pointed out. First, this design permits an on-line re-configuration of the number of bins, giving the possibility of having histograms with different resolutions. Moreover, the streaming paradigm is realized using a single clock source, thus permitting to run the system at the maximum allowable frequency, unlike state-of-the-art solutions in which a second faster clock source is used for the FSMs [8], [10].

The final *Histogram core* reported in Fig. 6 is the basis for successfully evaluating image histograms. Because of the streaming paradigm, the system has to be able to manage a continuous pixel flow, thus being able to handle multiple consecutive images. Considering the total latency of the *Histogram core*, the circuit is capable to process an image in a period equal to $T_f + L_{total}$, where T_f is the frame period (see the time diagram in Fig. 7). The *Histogram core* cannot consider a new image until the previous one is completely processed, thus limiting again the real-time capabilities of the system. To avoid this problem it is possible to duplicate the histogram core

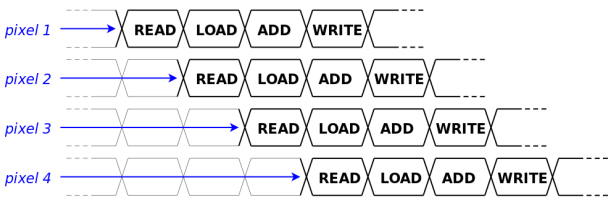

Figure 5: Streaming pixel flow.

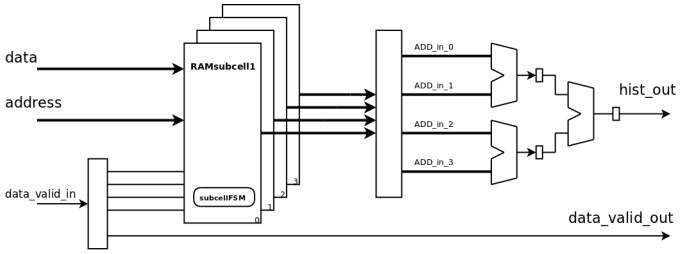

Figure 6: Histogram core.

as described in Fig. 8. In such a situation a new FSM, called *Store FSM*, directs the current image to the available active Histogram cell instance thus enabling a continuous histogram extraction.

III. Performance Evaluation

The proposed histogram core has been first tested on a cycle accurate Verilog simulator [11], then it has been synthesized on a mid-range FPGA. We adopted the Terasic DE0-nano development board [12] which embeds an Altera Cyclone IV FPGA with 22k Logic Elements (LE), 600 kbits on-board memory, and 144 9x9 bits DSP modules for the final implementation. A 1.3MPixels Omnivision CMOS Camera [13] was used as a real video source.

A. Algorithm footprint

As previously described, the proposed solution has the capability to re-configure at run-time the number of bins N, after having fixed at compilation-time the allocated memory M (in bits) and the bin resolution R (in bits). It is possible to vary N under according to the following equation:

$$N \leq \left\lfloor \frac{M/8}{R} \right\rfloor \tag{2}$$

where the factor 8 derives from the instantiation of 8 histogram subcells (in Fig. 2) in the definition of the Histogram core of Figs. 4 and 8. For instance, considering to allocate a memory array of 65536 bits (i.e., 8192 Bytes) and to set a resolution of 32 bits, it is possible to configure the core for a histogram with a maximum of 256 levels.

To keep a memory constrained design, the circuit has to be carefully sized especially in terms of the amount of the instantiated memory. Thus in the scenario of [3], let us suppose to have K histogram-based algorithms, such that each one

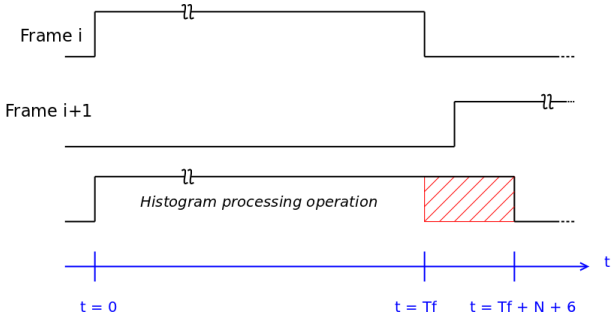

Figure 7: Time diagram.

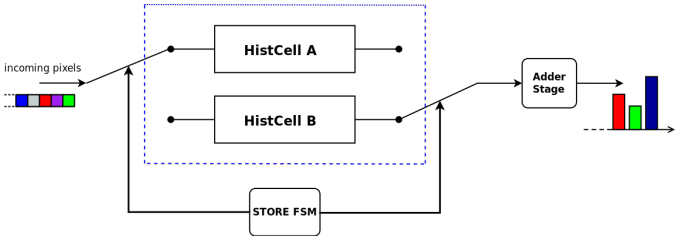

Figure 8: Standard image histogram core scheme.

requires a number of bins equal to N_i, with $i \in \{1...K\}$. The memory to be instantiated at compilation-time equals to:

$$M = 8 \cdot MAX(N_i) \cdot R. \quad (3)$$

B. Timing

As described in Eq. 1, the latency introduced by the Histogram core of Fig. 6 directly depends on the number of bins. The latencies introduced by the above mentioned core are detailed in Tab. I as a function of the number of bins N.

N (#bins)	Latency (Clock cycles)
8	14
16	22
32	38
256	262

Table I: Histogram core latencies as a function of N.

It must be pointed out as the total latency is completely independent of the image resolution expressed as number of pixels. By considering the final implementation histogram core design, which uses only 4% of the Altera Cyclone IV resources, and a clock frequency of 100 MHz, a VGA video stream can be processed at a rate of 260 frame per seconds.

C. Performance comparison with state-of-the-art

In this subsection the streaming based approach proposed in the paper is compared with respect to two state-of-the-art solutions. More in detail, the comparison is with [7], where a histogram is implemented making use of a linear array of bin-cells in which the computation proceeds in a pipelined fashion. We also compared our approach with [8], where the same approach of [7] is optimized using the *C-slow retiming* technique, thus reducing the critical path delay into the feedback loops, and improving the execution frequency. Both the considered solutions have been implemented in consumer Xilinx FPGAs.

Tab. II shows the allowed working frequency and the number of Logic Elements (LEs) used in each implementation. As far as system frequency is concerned, the proposed work, implemented in the Altera Cyclone IV EP4CE22 FPGA, is capable to process histograms with a maximum system clock over 100 MHz. The comparison in number of LEs between different chip vendors has been done by making use of the converter in [14]. The proposed work overcomes the state-of-the-art solutions with at least a factor 2, in terms of FPGA resource occupancy.

Moreover, the memory footprint and the latency are compared in Tab. III. Although the proposed work uses a bigger amount of memory with respect to [7], its memory footprint is half of the implementation proposed in [8]. However, for a fair comparison, it should also be taken into account the reconfigurability feature of the proposed solution. Both of the considered state-of-the-art works are not reconfigurable because of their architectural design, while the proposed approach can extract histograms with a configurable number of bins up to N. Finally, the latency of the proposed solution is lower than the compared works for all $N > 6$. In fact, both compared techniques have to wait N clock cycles to end the processing of the last pixel, and additional N clock cycles to empty the histogram structure.

	Frequency [MHz]	LEs [#]
System as in [7]	144	2422
System as in [8]	238	≈ 4800
Proposed solution	> 100	850

Table II: Allowed frequency and LEs occupancy comparison.

	Memory [bits]	Latency [clock cycles]
Realization in [7]	$N \times R$	$2N$
Realization in [8]	$8 \times N \times R$	$2N$
Proposed solution	$4 \times N \times R$	$N + 6$

Table III: Memory footprint and latency comparison.

IV. Conclusion

A reconfigurable parallel histogram core for real-time image processing in FPGA based low-cost smart cameras is presented in this paper. The main feature of the proposed core is the ability of working according to the streaming paradigm, while providing a full compatibility with the reconfigurable SC architecture presented in [3]. The core performance, obtained through a real implementation in the Altera Cyclone IV chip, shows the benefits of the proposed solution in terms of working frequency, LEs occupancy, and required memory, with respect to state-of-the-art solutions. The final implementation of the histogram core is able to process a 260 frame per seconds VGA video stream, thus enabling a wide range of applications in the smart camera networks scenario.

References

[1] B. Rinner, T. Winkler, W. Schriebl, M. Quaritsch, and W. Wolf, "The evolution from single to pervasive smart cameras," in *Proceedings of ACM/IEEE International Conference on Distributed Smart Cameras*, 2008, pp. 1–10.

[2] C. Salvadori, D. Makris, M. Petracca, J. Martinez del Rincon, and S. Velastin, "Gaussian mixture background modelling optimisation for micro-controllers," in *Advances in Visual Computing*, vol. 7431 of *Lecture Notes in Computer Science*, pp. 241–251. Springer Berlin Heidelberg, 2012.

[3] L. Maggiani, C. Salvadori, M. Petracca, P. Pagano, and R. Saletti, "Reconfigurable fpga architecture for computer vision applications in smart camera networks," in *Proceedings of ACM/IEEE International Conference on Distributed Smart Cameras*, 2013.

[4] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, vol. 1, pp. 886–893.

[5] M.U. Chowdhury, R. Rahman, J. Sana, and S.M.R. Kabir, "Fast scene change detection based histogram," in *Proceedings of IEEE/ACIS International Conference on Computer and Information Science*, 2007, pp. 229–233.

[6] P.R.R. Hasanzadeh, A. Shahmirzaie, and A.H. Rezaie, "Motion detection using differential histogram equalization," in *Proceedings of IEEE International Symposium on Signal Processing and Information Technology*, 2005, pp. 186–189.

[7] J.O. Cadenas, R.S. Sherratt, P. Huerta, and Wen-Chung Kao, "Parallel pipelined array architectures for real-time histogram computation in consumer devices," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1460–1464, 2011.

[8] J. Cadenas, R.S. Sherratt, P. Huerta, Wen-Chung Kao, and G.M. Megson, "C-slow retimed parallel histogram architectures for consumer imaging devices," *Consumer Electronics, IEEE Transactions on*, vol. 59, no. 2, pp. 291–295, 2013.

[9] B. Guthier, S. Kopf, M. Wichtlhuber, and W. Effelsberg, "Parallel algorithms for histogram-based image registration," in *Proceedings of International Conference on Systems, Signals and Image Processing*, 2012, pp. 172–175.

[10] E. Garcia, "Implementing a histogram for image processing applications," Tech. Rep., Xilinx Xcell Magazine, 2000.

[11] ModelSim, "ModelSim HDL simulator," http://www.mentor.com/products/fpga/model, 2013.

[12] Terasic Technologies Inc., "DE0-nano," www.terasic.com, 2012.

[13] OmniVision Technologies Inc., "OV9650 datasheet," www.dragonwake.com/download/arm9-download/OV9650.pdf.

[14] Altera, "Device comparison," http://www.altera.com/cgi-bin/device_compare.pl, 2013.