AN FPGA-BASED IMPLEMENTATION OF SPATIO-TEMPORAL OBJECT SEGMENTATION

Kumara Ratnayake and Aishy Amer

Concordia University, Electrical and Computer Engineering,
Montreal, Quebec, Canada
Email: [k_ratnay, amer]@ece.concordia.ca

ABSTRACT

This paper proposes a robust real-time, scalable and modular Field Programmable Gate Array (FPGA) based implementation of a spatio-temporal segmentation of video objects. The goal of this work is to translate an existing object segmentation algorithm into hardware to achieve real-time performance. The proposed implementation achieved an optimum processing speed of 133 MPixels/s while utilizing minimal hardware resources. The design was successfully simulated, synthesized and tested for real-time performance on an actual hardware platform which consists of a frame grabber with a user programmable FPGA - Xilinx Virtex-II Pro.

Index Terms— Image segmentation, Field programmable gate arrays, Video signal processing

1. INTRODUCTION

Object segmentation plays a key role in many video processing applications such as surveillance or machine vision. However, the computational complexity involved in object segmentation makes it difficult to achieve real-time performance on a general purpose CPU or DSP. There exists three main architectural approaches to this challenge 1) Application Specific Integrated Circuit (ASIC), 2) parallel computing, and 3) FPGAs. Evolving high density FPGA architectures such as those with embedded multipliers, memory blocks and high I/O (input/output) pin count make FPGAs an ideal solution in video processing applications [1, 2, 3].

The work in [4] demonstrates how a number of image segmentation algorithms can be implemented on FPGAs. The dynamic reconfigurability feature of the FPGAs allows to reconfigure a part or complete FPGA within a fraction of a microsecond, and the paper shows how multiple image processing algorithms can be sequentially applied to the image by using the same FPGA.

Another FPGA implementation for segmenting text in images is in [5]. Experimental results show that this algorithm implemented in FPGA achieved a speedup of close to 250 compared to a general purpose CPU implementation. However, this implementation runs at 5 MHz which is well below the real-time performance.

The study in [6] partially involves FPGA-based implementation of image segmentation based on the resistive-fuse network model.

An extensive comparison between FPGA and DSP implementations of image classifier for object detection is in [7]. Although the performance of the FPGA implementation significantly overpasses that of the DSP implementation, its performance and scalability is heavily limited and embedded by the hardware platform chosen.

THIS WORK WAS SUPPORTED, IN PART, BY THE *FONDS DE LA RECHERCHE SUR LA NATURE ET LES TECHNOLOGIES DU QUEBEC* (NATEQ).

In this paper, we propose a real-time, scalable and compact architecture for FPGA-based object segmentation. Section 2 describes the object segmentation used and Section 3 proposes our architecture. Section 4 contains verification and synthesis results and Section 5 concludes this paper.

2. OVERVIEW OF THE REFERENCE SPATIO-TEMPORAL OBJECT SEGMENTATION ALGORITHM

An object segmentation method categorizes homogeneous pixels of a frame into a region. Video object segmentation methods [8] can be classified based on their automation, spatial accuracy, temporal stability, and computation load. Computationally expensive methods give, in general, accurate results while low-computation methods may fail. However, few of the methods are tested on a large number of videos, throughout long videos, on noisy videos, and without parameter tuning. We select the non-parametric segmentation method in [9] due to its low computation and noise and temporal stability. These features forgo spatial accuracy, e.g., at object boundaries. Such a method is most appropriate to applications, e.g., video surveillance, where stability under varying conditions is of more concern than accurate object boundaries. Furthermore, [9] is well suited for hardware implementation such as a modern FPGA due to its modularity, simplicity and reduced resources requirements. Fig.1 illustrates the block diagram of the method [9] which consists of three main modules: motion detection, spatio-temporal thresholding, and morphological edge detection.

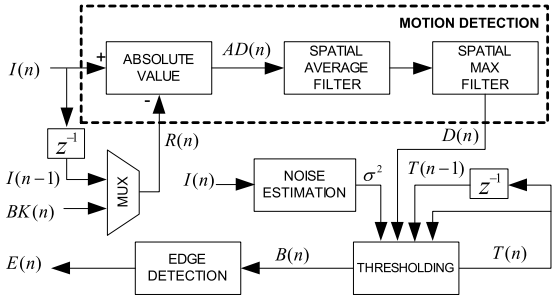

Fig. 1. Block Diagram of the Object Segmentation [9].

The motion detection finds first the absolute frame difference, $AD(n)$ at time instant n, between the current $I(n)$ and a reference frame $R(n)$. $R(n)$ can be either a background $BK(n)$ or the previous frame $I(n-1)$ in a video sequence. $AD(n)$ is then spatially filtered by both an average and a max filter.

In the thresholding module, a global spatial threshold T_g is first computed as follows. The spatially-filtered frame $D(n)$ is divided into K consecutive non-overlapping blocks, $W_k, k \in \{1, ..K\}$. The histogram of each W_k is split into L equal sections. The most-frequent gray-level g_{pl} of each histogram section is found and

$$T_g = \frac{\sum_{k=1}^{K} (\lambda_l + \mu_k)}{K.L + K} \quad (1)$$

where $\lambda_l = \sum_{l=1}^{L} g_{pl}$ and μ_k is the pixel average of W_k. Note that T_g is obtained using block local and global data. T_g is then proportionally adapted to the noise variance σ^2 using

$$T_\varsigma = T_g + a.\sigma^2 \quad (2)$$

where $0 < a < 1$ and σ^2 is estimated using [10]. This noise-adapted T_ς is then quantized to maintain spatio-temporal stability where quantization down to three levels yields good results [9]. The quantized threshold T_q is passed through a memory system that holds the threshold of the previous frame and determines the threshold $T(n)$ based both on new quantized threshold T_q as well as the previous threshold $T(n-1)$. Finally, $D(n)$ is globally thresholded by $T(n)$ creating a binary frame $B(n)$.

To extract object boundaries, the edges $E(n)$ in $B(n)$ are detected in the morphological edge detection module. Here, a 2x2 square kernel is moved over the entire $B(n)$ and if the result of Boolean AND operation on these four binary pixels is false, then the output pixels are set to white if their corresponding pixels in the input frame are white, otherwise output pixels are set to black [9]. In [9], $E(n)$ is passed through a contour tracing and filling algorithm to label the objects, and this step is not implemented in this paper.

3. PROPOSED PIPELINED ARCHITECTURE

The overall system level architecture of the FPGA design is illustrated in Fig. 2. It consists of a Direct Memory Access (DMA) module and three main processing blocks for motion detection, spatio-temporal thresholding and morphological edge detection.

3.1. Proposed DMA Architecture

An efficient management of data transfers within a system is the key to any real-time hardware implementation. In our implementation, we designed a scalable and versatile DMA architecture that can be easily configured by a simple set of registers. The proposed DMA consists of 4 KB deep First In First Out (FIFO) memories connected to each read and write DMA channels, a DMA controller (DMACTLR) to manage these FIFOs, and a DDR memory controller. A write transfer to the memory is initialized by filling the corresponding write FIFO (up-to a maximum of 2 KB), and sending a request to DMACTLR. Whenever a read FIFO is half empty, a read request is automatically initialized. An internal cache memory is used to store the addresses and transfer descriptions of each DMA channel. A round-robin arbitrator arbitrates all parallel requests from each channel and serves the selected DMA channel.

Our architecture for motion detection and the DMA is scalable in that motion detection can be configured into the two modes (background $BK(n)$ and previous $I(n-1)$ frame) on-the-fly. In the former case, the DMA is programmed to store the acquired frame as the background frame in the memory and it continuously read the background frame and sends it to the motion detection module along with $I(n)$. In the later case, the DMA transfers newly arrived $I(n)$ to the memory for future processing as well as to the motion detection module. At the same time, the DMA reads $I(n-1)$ that was stored in the memory (during the last frame time) and sends it to the motion detection module. The output frame $D(n)$ of the motion detection is routed back to the memory and to the spatio-temporal thresholding node. The spatio-temporal thresholding block takes a full frame time to compute a threshold, hence it is necessary to buffer the frame being processed in the memory until a valid threshold is available. Within this duration, the previous motion-detected frame is read from the memory and is sent to the last processing block for morphological edge detection. The proposed DMA architecture manages all these massive data parallelism in such a way that is seamless to any of the processing blocks.

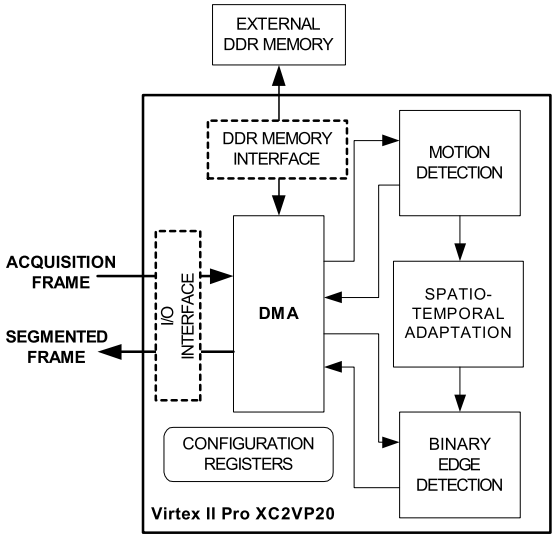

Fig. 2. System-Level Architecture of Object Segmentation.

3.2. Scalable Motion Detection Implementation

The absolute difference frame $AD(n)$ is computed by a simple subtractor and its absolute value is routed to the spatial average and max filters. We architectured the spatial filters to be flexible and scalable in number of ways: 1) our implementation can change the size of the both filters from any configuration between 1x1 and 5x5 on-line, and 2) the frame resolution is programmable allowing to support different video cameras. The architecture is designed in a modular manner, so that future design expansions can be easily feasible. For instance, if the design has to support a video camera with more than 2 KB line width, it can be achieved by using multiple instances of the existing modules. We also minimized the memory bandwidth that would require to write and read previous lines for two-dimensional filters by using internal BRAMs as line buffers.

3.3. Spatio-Temporal Thresholding Architecture

The high-level architecture designed for the spatio-temporal thresholding is shown in Fig. 3. The novelty of this architecture is that it does not require any external memory to extract the individual blocks. The block extractor (BE) splits the motion-detected data into M vertical blocks, which are then fed into M Intensity Histogram

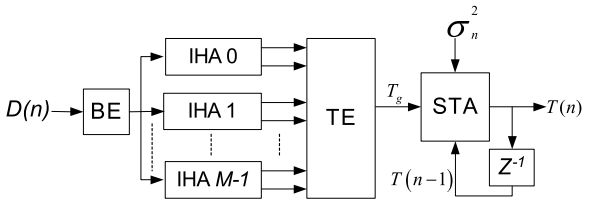

Fig. 3. High-Level Architecture of Spatio-Temporal Threshold.

Analysis (IHA) modules. Each IHA generates μ_k and λ_l for the corresponding block. The Threshold Estimator (TE) takes those values to produce T_g for Spatio-Temporal Adaptation (STA) module. The STA consists of an adder that adds T_g to a weighted value of noise variance to get T_ς, and two priority encoders. The first encoder produces T_q by quantizing T_ς down to three quantization levels which are defined with three user programmable registers. The second priority encoder selects $T(n)$ according to T_q and $T(n-1)$.

3.3.1. Architecture of the IHA Module

The IHA consists of two main processing nodes - Intensity Average and Histogram Analysis which compute μ_k and λ_l respectively, and a Controller and an Address Generation unit, that generates the signals required to control these processing nodes. The overall architecture is shown in Fig. 4. In the Intensity Average module, we

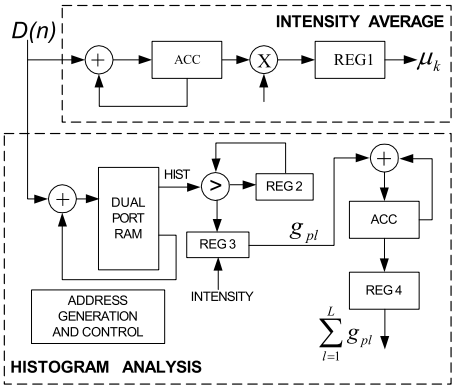

Fig. 4. Architecture of Intensity Histogram Analysis Module.

used a multiplier as a divider to obtain the average value. Hence, the resources usage is minimal and the result of the average is obtained with less pipelined delay when compared to a pure divider usage. Histogram Analysis block first calculates the histogram of the input frame using a BRAM and an adder. After the entire frame data for a $W \times H$ block is entered, the histogram will be available in the BRAM. When the histogram is sequentially read, Reg 2 holds the maximum value within an interval $l, l \in \{1, ..L\}$, and Reg 3 keeps the corresponding gray value, g_{pl}. Once the complete histogram is read, g_{pl} is accumulated over the entire intervals, and the result of λ_l will be stored in the Reg 4.

3.3.2. Architecture of the TE Module

The architecture of the threshold estimator is shown in Fig. 5. The inputs, μ_k and λ_l, to the TE block arrive in serially. This allows us to use two multiplexers to select the appropriate operands to the

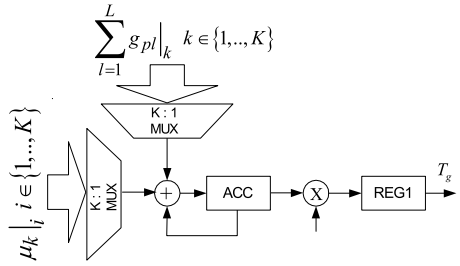

Fig. 5. Threshold Estimator Architecture.

accumulator, which minimizes the resources usage. After all the data of an entire frame has arrived, T_g, will be available in the REG1.

3.4. Morphological Edge Detection Architecture

The architecture of the morphological edge detection is shown in Fig. 6. The Morphological Engine (ME) evaluates the Boolean

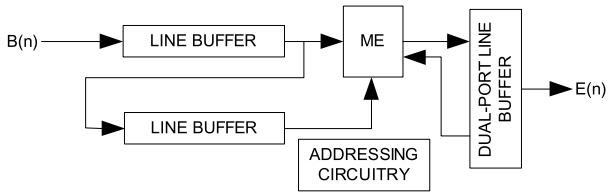

Fig. 6. Circuitry for Morphological Edge Detection.

condition (see Section 2). On the output, we configured an internal BRAM as a Dual Port Line Buffer (DPLB) to keep track of the partially estimated edge frame, hence ME can access and modify the content of the DPLB without degrading its access time. Once ME has accessed and modified the entire line in the DPLB twice, the content of the DPLB is $E(n)$. This is read from DPLB and sent to the DMA to transfer to the output port of the FPGA.

4. DESIGN VERIFICATION AND SYNTHESIS

4.1. Verification

We simulated the proposed design with a video sequence, "Hall" (300 frames of 352 pixels x 288 lines) to thoroughly verify the integrity of our implementation. We used a background frame as a reference (see Fig. 1). Fig. 7 is an example result obtained with the FPGA simulation and the reference C software implementation which subjectively reveals that both results are closely identical.

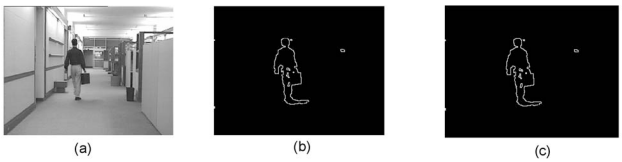

Fig. 7. (a) 54th frame in the captured video sequence, (b) segments with the reference C implementation, and (c) FPGA segments.

In addition, we used two objective measures to compare our implementation results $E_{hw}(n)$ with the reference C results $E_{sw}(n)$.

Product of Correctly Classified Proportions, PCP is a widely used objective measure to evaluate binary images [11]. Here $E_{sw}(n)$ serves as the ground-truth data. We can see in Fig. 8(a) that $E_{hw}(n)$ is extremely close to $E_{sw}(n)$. Notice that if PCP = 1, then both results are identical ($E_{hw}(n) = E_{sw}(n)$) and if PCP = 0, then they are inverse ($E_{hw}(n) = \overline{E_{sw}(n)}$). We also computed the sum of the pixels that are different between the results of the proposed implementation and reference C implementation as $\Delta_{hw} = \sum |E_{hw}(n) - E_{sw}(n)|$, and Fig. 8 (b) shows that maximum of Δ_{hw} is 15 pixels.

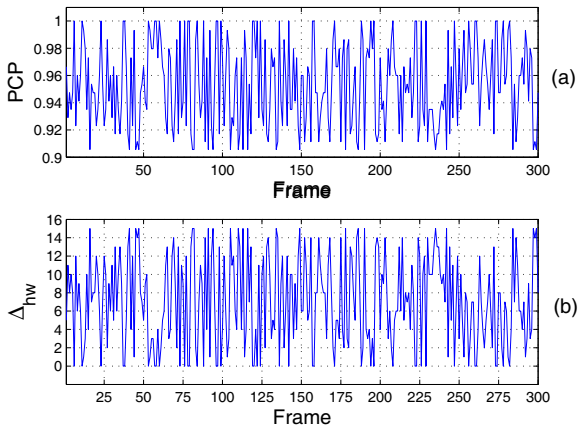

Fig. 8. (a) Comparison between software and hardware implementations with PCP objective measure [11], and (b) difference of total pixels between software and hardware implementations Δ_{hw}.

We have also successfully tested the accuracy and performance of the proposed implementation on an actual FPGA of a frame grabber using a high-speed camera.

4.2. Synthesis and FPGA Implementation

We have designed and simulated the proposed FPGA architecture for the spatio-temporal object segmentation algorithm in VHDL and synthesized with Synplicity Synplify 7.3. The synthesized design was then placed and routed for XC2VP20 FPGA with Xilinx ISE 7.1 Alliance tool. The implemented design occupies approximately 60% of the area of an XC2VP20 FPGA (37 % of slices, 27 % of LUTs (look up tables), 59 % of BRAMs, and 9 % of multipliers). Xilinx XPower tools estimated the power dissipation of the implementation to be less than 2 W for a toggle-rate of 50 % . The design is easily able to run up to 133 MHz, which means that it takes only 7.5 ms to complete segmentation of 1024x1024 frame (including input frame load and result frame unload timing), which is more than sufficient for current and near future video processing applications.

4.3. Comparison to the Existing Methods

The architecture presented in [5] runs at 5 MHz and it takes 360 ms to segment a 1024x1024 frame. In contrast, our architecture achieved a significant higher clock rate of 133 MHz, hence it takes only 7.5 ms to complete segmenting a 1024x1024 frame, including input read and output write timing. Moving data between memory and FPGA affects the scalability and the overall performance of an implementation. Our versatile DMA architecture is more generic and scalable than the data movement procedure presented in [7].

5. CONCLUSION

This paper proposed a novel, robust, scalable and modular FPGA architecture for real-time spatio-temporal object segmentation. We used advanced design techniques such as heavy pipelining and data parallelism, hence achieved an optimal speed of 133 MHz while utilizing minimal hardware resources. Furthermore, our architecture avoids many re-designing efforts by its inherent scalability and adaptivity, for instance, many of the algorithm specific parameters such as spatial filter size can even be programmed on-the-fly.

The proposed architecture can process one pixel per processing clock cycle (7.5 ns); i.e., it has an overall throughput of 133 MPixels/s. This processing speed can be increased up to a factor of 8, by modifying the processing nodes to process multiple pixels (up to 8) in one clock cycle. In order to adapt to noise automatically, the design can be integrated with a noise estimation implementation.

6. REFERENCES

[1] C. T. Huitzil and M. A. Estrada, "Real-time image processing with a compact FPGA-based systolic architecture," *Elsevier Journal of Real-time Imaging*, vol. 10, pp. 177–187, 2004.

[2] C. Rambabu, I. Chakrabarti, and A. Mahanta, "An efficient architecture for an improved watershed algorithm and its FPGA implementation," *In Proc. IEEE International Conference on Field-Programmable Technology*, pp. 370–373, Dec 2002.

[3] K. V. Asari, T. Srikanthan, S. Kumardemi, and D. Radhakrishnan, "A pipelined architecture for image segmentation by adaptive progressive thresholding," *Journal of Microprocessors and Microsystems*, vol. 23, pp. 493–499, 1999.

[4] D. Demigny, L. Kessaland, R. Bourguiba, and N. Boudouani, "How to use high speed reconfigurable FPGA for real time image processing," *in Proc. IEEE Conference on Computer Architecture for Machine Perception*, pp. 240–246, 2000.

[5] N.K. Ratha, A.K. Jain, and D.T. Rover, "FPGA-based high performance page layout segmentation," *Proceedings of the 1996 Great Lakes Symposium on VLSI*, pp. 29–34, Mar 1996.

[6] T. Nakano, T. Morie, and A. Iwata, "A face/object recognition system using FPGA implementation of coarse region segmentation," *SICE 2003 Annual Conference*, vol. 2, pp. 1552–1557, 2003.

[7] P. McCurry, F. Morgan, and L. Kilmartin, "Xilinx FPGA implementation of an image classifier for object detection applications," *International Conference on Image Processing*, vol. 3, pp. 346–349, 2001.

[8] D. S. Zhang and G. Lu, "Segmentation of moving objects in image sequence: A review," *Springer Circuits, Systems and Signal Processing (Special Issue on Multimedia Communication Services)*, vol. 20, pp. 143–183, 2001.

[9] A. Amer, "Memory-based spatio-temporal real-time object segmentation," *in Proc. SPIE Int. Symposium on Electronic Imaging, Conf. on Real-Time Imaging (RTI)*, vol. 5012, pp. 10–21, Jan 2003.

[10] A. Amer and E. Dubois, "Fast and reliable structure-oriented video noise estimation," *in Proc. IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, pp. 113–118, Jan 2005.

[11] P. L. Rosin, "Thresholding for change detection," *Computer Vision and Image Understanding*, vol. 86, pp. 79–95, 2002.