SEQUENTIAL, IRREGULAR AND COMPLEX OBJECT CONTOUR TRACING ON FPGA

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 real-time, robust, scalable and compact Field Programmable Gate Array (FPGA) based architecture and its implementation of contour tracing of video objects. Achieving real-time performance on general purpose sequential processors is difficult due to the heavy computational and memory access demands in contour tracing, thus a hardware acceleration is inevitable. Our finding to the existing related work confirms that the proposed architecture is much more feasible, cost effective and features important algorithmic-specific qualities, including deleting dead contour branches and removing noisy contours, which are required in many video processing applications. Our implementation achieved an optimum processing clock of 158 MHz while utilizing minimal hardware resources and power. The proposed FPGA design was successfully simulated, synthesized and verified for its functionality, accuracy and performance on an actual hardware platform which consists of a frame grabber with a user programmable Xilinx Virtex-4 SX35 FPGA.

Index Terms— Field programmable gate arrays, Object detection, Image edge analysis, Video signal processing

1. INTRODUCTION

Contour tracing is a method that links connected neighborhood pixels in a binary edge frame, which is becoming increasingly important in many image and video processing applications such as video surveillance [1], object based video coding, e.g., MPEG-4 and MPEG-7, [2], pattern recognition and computer vision.

Considerable efforts have been given to devise contour tracing algorithms on software platforms [3, 4, 5]. However, the computational complexity involved in contour tracing makes it difficult to achieve real-time performance on general purpose sequential processors such as CPU or DSP. Hence, hardware accelerations are necessary to attain real-time contour tracing. Emerging FPGAs with embedded multipliers, memory blocks and high pin counts are increasingly employed on hardware platforms in many video processing applications due to their run-time reconfigurability and lower development cost when compared with other architectural approaches such as full custom Application Specific Integrated Circuits (ASICs).

In this paper, we propose a real-time, scalable and compact FPGA-based architecture and its implementation of contour tracing, which is an extension to our previous art on FPGA-based video segmentation presented in [6]. The rest of the paper is organized as follows: Section 2 describes the related work and Section 3 gives an overview of the contour tracing algorithm [3]. Our proposed architecture is outlined in Section 4. Section 5 contains experimental results while Section 6 concludes this paper.

This work was supported, in part, by the *Fonds de la recherche sur la nature et les technologies du Quebec (NATEQ).*

2. RELATED WORK REVIEW

Chia et al. [7] proposes a parallel VLSI architecture which consists of $N+1$ processing elements for generating the chain codes of object contours in a binary frame with N raws. The algorithm proposed in [7] can complete contour extraction in $3N$ cycles, assuming the input binary frame is already stored in memory. However, in order to complete tracing in $3N$ cycles, [7] requires simultaneous reading of all N raws and simultaneous writing of chain codes to memory. Moreover, final contours are generated by accessing memory in a random fashion. Contour tracing methods inherently involves random data movements between memory and contour extraction unit(s). Thus, performance and feasibility of implementing a given contour tracing architecture or algorithm depend heavily on the efficiency of the memory and the robustness of the memory data accessing mechanism. Hence, as presented in [7], the architecture is virtually infeasible to implement with presently available memories.

A full custom VLSI CMOS design for extracting contours is presented by Agi et al. in [8]. Here, authors attempt to minimize the memory usage by partitioning the input frame into smaller regions and distributing these regions to an array of processing elements (PEs). Each PE in [8] consists of its own memory and a contour tracing unit, and uses a 2x2 window for extracting partially completed contour lists. However, [8] fails to produce completed contour tracing, unless a full object can be completely stored in the relatively small processing memory.

Moreover, the conventional contour tracing algorithms used in [7] and [8] do not have the intelligent features present in [3] method and subsequently in our proposed implementation. These characteristics include detection and elimination of dead contour branches and noisy contours, which are required in many video processing applications.

3. OVERVIEW OF THE REFERENCE CONTOUR TRACING ALGORITHM

The gaps free edge image, $E(n)$, produced with spatio-temporal motion detection followed by morphological edge detection [6, 1] consists of object contours (white points, p_w) and a background (black points, p_b). The goal of a contour tracing algorithm is to link the white points, p_w, into a group. Unlike conventional contour tracing algorithms, [3] extracts contours of all closed complex objects while deleting dead or inner branches, and excluding contours of noisy objects. The detection and exclusion of such contours are important and necessary in video surveillance and other video process-

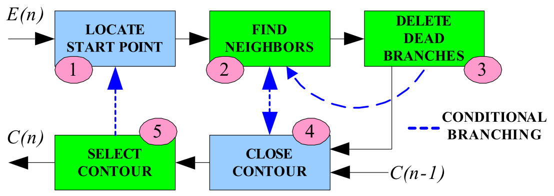

Fig. 1. Contour Tracing Algorithm [3].

ing applications, hence, we select [3] for our FPGA implementation. However, inherent sequential nature of [3] brings some challenges to its implementation on parallel hardware devices such as FPGAs. In addition, the process of excluding a contour in [3] requires manipulating previous frame contours. As such, an efficient method of retrieving appropriate contours of previous frame is needed. We propose an efficient cache architecture, in the FPGA, to overcome the sequential issue and a robust mechanism to access previous frame contours stored in a memory without interfering the core processing units.

A block diagram of the tracing method [3] is depicted in Fig. 1. More detailed description of this referenced algorithm can be found in [3]. As can be seen from Fig. 1, the algorithm can be partitioned into five sub modules, which are briefly outlined next.

Locating A Start Point (Rule 1): The edge image, E(n), is scanned in raster mode (from left to right and from top to bottom) until an unvisited white points, p_w, which has at least one unvisited neighbor is found. If such a p_w exists, then the set starting point, p_s, $= p_w$, set the current point, p_c, $= p_s$, and perform Rule 2.

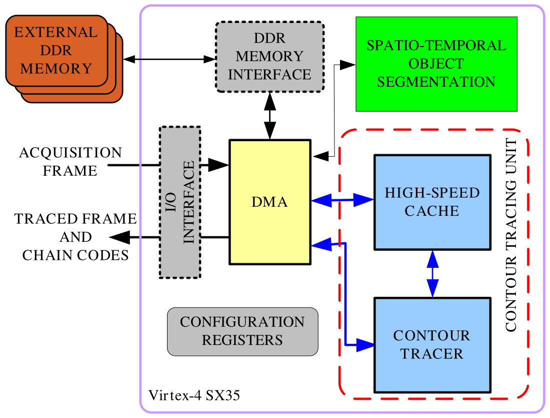

Fig. 2. System-Level Architecture of Contour Tracing.

Finding the Rightmost Neighbor Points (Rule 2): In [3], tracing is performed in anti-clockwise direction searching for a rightmost neighbor of the current point in an 8-neighborhood, p_i. The current searching direction, d_s, is defined by the direction from the previous point, p_p, to the current point. The direction to the rightmost neighbor of a current point which depends on d_s is formulated as:

$$\{d_s + 6 + [(d_s + 1)\ mod\ 2]\}\ mod\ 8. \qquad (1)$$

Eq. 1 states that the algorithm searches up-to five and six rightmost neighbors for even and odd value of d_s, respectively. If a rightmost neighbor is present, then p_c is labeled visited, $p_p = p_c$, $p_c = p_i$, and Rule 4 is executed, otherwise p_c is a dead branch and deleted by performing Rule 3.

Deleting Dead Branches (Rule 3): Rule 3 eliminates current point p_c from $E(n)$, sets $p_c = p_p$ and p_p to its previous neighbor, and finally activates Rule 2.

Contour Closing (Rule 4): If $p_c = p_s$ or p_c is labeled visited, the current contour, C_c, being traced is closed. In both cases, Rule 4 removes all the points of the C_c from $E(n)$ and perform Rule 5. Furthermore, if p_c is labeled visited, remaining dead points of C_c are eliminated from $E(n)$. If C_c is not close, then Rule 4 stores coordinate and the chain code of p_c and activates Rule 2.

Contour Selection (Rule 5): In this rule, C_c is verified with three measures before adding C_c to the contour list, $C(n)$. C_c is not added to $C(n)$ if 1) current contour length, P_c, is too small, or 2) P_c is small and has no corresponding contour in the previous contour list, C_{n-1}, or 3) C_c resides in an already traced contour, C_p, causing a low spatial homogeneity of the object of C_p.

Although [3] records contours in both the chain code and the point coordinates, we use chain code in our implementation since the chain code requires less memory for storage.

4. PROPOSED ARCHITECTURE

Fig. 2 exemplifies our proposed system level architecture of contour tracing. We have also incorporated our previous work on FPGA-based object segmentation [6] as a front-end processing engine for our proposed architecture. The proposed contour tracing architecture consists of a HIGH-SPEED CACHE and a CONTOUR TRACER.

4.1. Architecture of HIGH-SPEED CACHE

The performance of the any sequential contour tracing architecture is heavily determined by the efficiency of the memory and its data transferring mechanism. The algorithm [3] demands reading a 3x3 window randomly from memory. Intuitively, Static-RAM (SRAM) devices are ideal for random access applications, however, they suffer from few significant requisites. First, SRAM is costly and it increases physical area and power consumption. Second, real-time precessing requires reading a 3x3 window as fast as possible, ideally in one clock cycle. SRAM needs 3 clocks (1 clock/1 raw) and accessing 3 bits in a raw deflates the SRAM bandwidth, as the data bus of conventional SRAM is significantly greater than 3 bits. Thus, we propose a scalable, efficient and high speed cache architecture by exploiting FPGA memory blocks (BRAMs), which is depicted in Fig. 3. Main attributes of our cache are: 1) simultaneous read and write of 16 pixels in each direction, 2) 4.8 GBits/s aggregate throughput and 3) scalability with $O(n)$ area complexity.

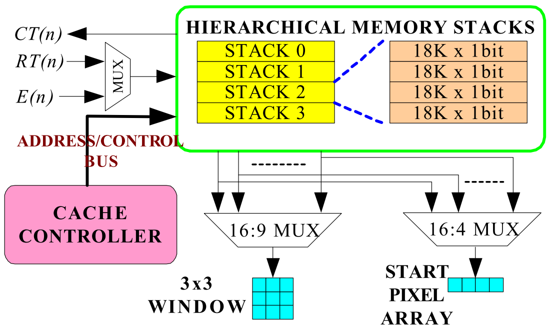

Fig. 3. Scalable Architecture of the HIGH-SPEED CACHE.

In order to complete contour tracing of one frame, cache is sequentially required to 1) store $E(n)$, 2) read START PIXEL ARRAY (SPA), 3) generate 3x3 WINDOW, 4) write reconstructed contour traced frame ($RT(n)$) and 5) read contour traced frame ($CT(n)$). Our proposed cache has four HIERARCHICAL MEMORY STACKS, HMS, which consists of four BRAMs constructed as dual port with 1 bit wide and 18K deep. We write the first line of $E(n)$ in HMS0, the second in HMS1,.., the fifth in HMS0 and so on. In the same sequence, we store four pixels in the four BRAMs of each HMS. The main motivation for storing in such a sequence is that it facilitates reading any 16 pixels in one clock cycle. The CACHE CONTROLLER, CACTRL, schedules $E(n)$, $CT(n)$, $RT(n)$, 3x3 WINDOW, and SPA by managing all addressing and read/write controls to each HMS and controlling the three MUXs. It can be easily shown that the total scheduling pipeline is $< 6T$ where $T = 0.7$ ms is the time requires to access a CIF frame at 150MHz clock.

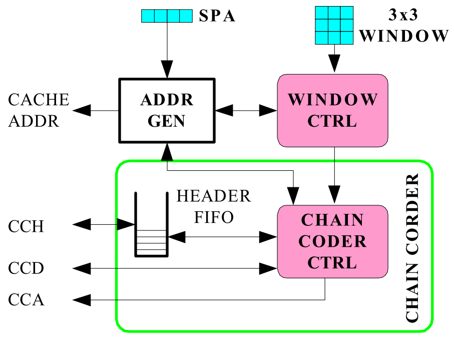

Fig. 4. Overall Schematic of the CONTOUR TRACER.

4.2. CONTOUR TRACER Architecture

As illustrated in Fig. 4, most of the functionality of the proposed CONTOUR TRACER are controlling various contour tracing events. ADDR GEN perform rule 1 of [3], and generates direct cache addresses, DCA, based on the pixels in SPA and controls signal received from the two controllers, WINDOW CTRL and CHAIN CODE CTLR. WINDOW CTRL takes 3x3 WINDOW and determines if the tracing rules 2-5 are valid by means of a Finite State Machine (FSM) and some trivial logic employed to evaluate rule 5.

As the the name implies, the CHAIN CODER produces Chain Code Streams (CCS) of the contours. We write CCS, while they are being produced, to the DDR memory. A CCS already written to the memory may not be valid if it is a dead branch or Rule-5 caused it to remove, therefor, such a CCS should be identified, and be excluded from the contour list. We adopted headers (CCH) and tails for CCS as well as for the Chain Code Frame (CCF) starting with 0x8 nibble as a marker followed by header descriptors. Notice that we intentionally use 4 bits for chain codes which are from 0x0 to 0x7, therefore, header marker 0x8 can be easily distinguished. Fig. 5 defines a complete chain code bit stream. When a CCS belongs to a dead branch, CHAIN CODER sets a flag in the CCH. On completion, of contour tracing of a full frame, CHAIN CODER extracts the header descriptors to remove any CCS of dead branches, and reconstructs contour traced frames $RT(n)$ in the cache. $RT(n)$ is transfered to the DMA along with the chain codes as the final output.

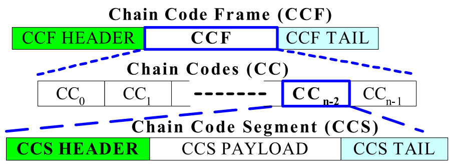

Fig. 5. Contour Bit Stream Structure.

4.3. Improved 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 [6] architecture that can be easily configured by a simple set of registers. Furthermore, the DMA constitutes the ability to access a memory location with an address provided by a processing unit, while paying special attention to minimize the performance overhead caused when a small amount of data is accessed from the memory.

5. EXPERIMENTAL RESULTS

5.1. Verification

We verify the integrity of the proposed design in conjunction with [6], by simulating the *Hall* video sequence, which consists if 300 frames of 352 pixels x 288 lines. The edge frames produced with the result of contour tracing of FPGA simulation and the reference C software implementation for the 54th captured frame in the *Hall* video sequence is shown in Fig. 6.

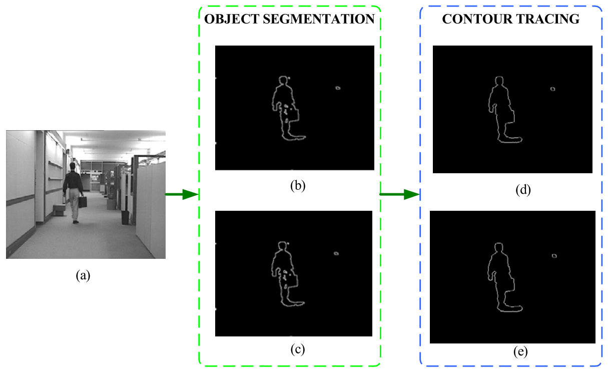

Fig. 6. Subjective comparison between the FPGA and software implementation results - (a) 54th frame in the captured video sequence. (b) Spatio-temporal object segmentation result with reference C [3], and FPGA implementation (c) [6]. (d) Contour tracing results with C, and proposed FPGA implementation (e).

To verify the result of our proposed FPGA implementation objectively with the software implementation, we used the Product of Correctly Classified Proportions [9], PCP, measure which is widely known and used as an objective measure for evaluating binary images. Serving software contour-traced frames $CT_{sw}(n)$ as the ground-truth data, Fig. 7 (a) shows that the PCP is close to 1 and always above 0.9 for the 300 frames of *Hall* video sequence. Notice that when a binary image is identical to the ground-truth frame, then PCP is 1. Thus, contour-traced frames produced by FPGA, $CT_{hw}(n)$, are very close, if not identical, to the $CT_{sw}(n)$.

Moreover, we enumerated the sum of the white pixels, Δ_{hw}, in the absolutely difference frames, $|CT_{hw}(n) - CT_{sw}(n)|$. Fig. 7 (b) exemplifies that $\Delta_{hw} \geq 21$ for *Hall* video sequence. Notice that the object segmentation results [6] already contribute a significant fraction in this Δ_{hw}.

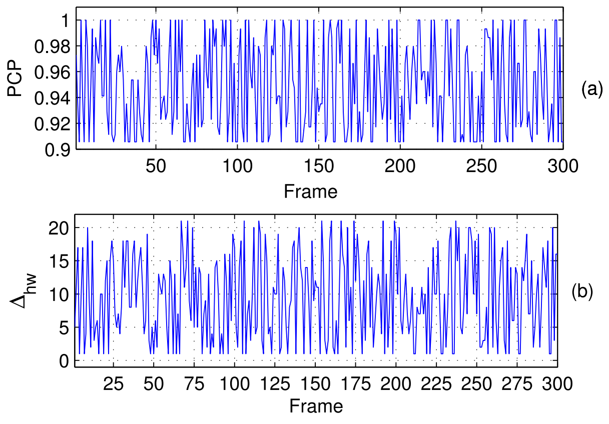

Fig. 7. (a) Comparison between software and hardware implementations with PCP objective measure [9], and (b) difference of total pixels between software and hardware implementations Δ_{hw}.

Furthermore, we have successfully verified the actual functionality, accuracy and performance of the proposed FPGA implementation on a frame grabber using a high-speed camera.

5.2. Synthesis and FPGA Implementation

We have coded and simulated the proposed architecture in VHDL, synthesized and implemented to a Xilinx Virtex-4 SX35 FPGA. The implemented architecture occupies 7% of registers, 4% of LUTs (look up tables), 11% of BRAMs, and 1% of multipliers of the FPGA. Our proposed-constrained implementation achived a clock rate of 158 MHz, and consumes 1.2 W for a toggle-rate of 50%. On the actual hardware platform, we set the processing clock in the FPGA to 150 MHz, which enabled the FPGA to capture and complete contour tracing at 238 frames/s (in 4.2 ms) for the CIF video resolution.

5.3. Comparison to the Existing Methods

The hardware architecture presented in [7] is fundamentally infeasible to implement due to its requirements of random and simultaneous memory access. Currently available memories do not possess greater than two read/write ports, but [7] requires a minimum of $N(>> 2)$ ports to read simultaneously N raws of a frame. Furthermore, additional simultaneous, random and fast accesses to the memory are required in [7] when each raw produces partially completed chain codes, and these are read and linked to create the final chain codes. As a result of this heavy memory access requirement, [7] needs an enormous amount of pins, which are not currently available even on the largest FPGA.

[8] fails to trace complete contours of large objects and requires an external post processor to link partially completed contours. Having an external post processor in addition to the core contour tracing circuitry increases cost, power, and physical area.

The architectural requirements needed in [7, 8] prevail having realistic hardware acceleration methods for contour tracing. In contrast, we exploited heterogeneous resources readily available on FPGA devices, adopted them in our architecture and devised a real hardware solution. Both [7, 8] need N contour processing units, whereas our method consists of only one tracing unit. Moreover, [7, 8] lack key contour tracing features such as deleting dead branches and removing noisy contours, and therefore, [7, 8] are not suitable for video applications such as video surveillance.

6. CONCLUSION

In this paper, we proposed a robust real-time, scalable and compact FPGA-based architecture and its implementation of contour tracing of video objects. Intentional use of heterogeneous resources in FPGAs, and advanced design techniques such as heavy pipelining and data parallelism enabled us to achieve an impressive contour tracing throughput of 238 frames/s, while consuming minimal power and resources. We verified our proposed implementation on an actual Virtex-4 SX35 FPGA platform for its functionality, accuracy and real-time performance. Existing methods lag behind our solution in key factors such as feasibility, cost and algorithmic-specific qualities, such as deleting dead contour branches and exclusion of noisy contours, which are required in many video processing applications.

Future work includes integrating with a noise estimation implementation in order to adapt automatically to video noise in our previous work on FPGA-based object segmentation.

7. REFERENCES

[1] A. Amer, "Voting-based simultaneous tracking of multiple video objects," *in IEEE Trans. Circuits and Systems for Video Technology*, vol. 15, pp. 1448–1462, Nov. 2005.

[2] H. Tsuji, S. Saito, H. Takahashi, and M. Nakajima, "Estimating object contours from binary edge images," *in Proc. IEEE Int. Conference on Image Processing (ICIP)*, vol. 3, pp. 453–456, Sep. 2005.

[3] 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.

[4] F. Chang, C.J. Chen, and C.J. Lu, "A linear-time component-labeling algorithm using contour tracing technique," *Computer Vision and Image Understanding*, vol. 93, pp. 206–220, Feb. 2004.

[5] T. Pavlidis, "Contour filling in raster graphics," *in Proc. ACM Annual Conference on Computer Graphics and Interactive Techniques*, pp. 29–36, Aug. 1980.

[6] K. Ratnayake and A. Amer, "An FPGA-based implementation of spatio-temporal object segmentation," *in Proc. IEEE Int. Conference on Image Processing (ICIP)*, pp. 3265–3268, Oct. 2006.

[7] T. L. Chia, K. B. Wang, L..R. Chen, and Z. Chen, "A parallel algorithm for generating chain code of objects in binary images," *Information Sciences Informatics and Computer Science*, vol. 149, no. 4, pp. 219–234, Feb. 2003.

[8] I. Agi, P. J. Hurst, and A. K. Jain, "A VLSI processor for parallel contour tracing," *in Proc. IEEE Transactions on Signal Processing*, vol. 40, no. 2, pp. 429–438, Feb. 1992.

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