Real-Time Canny Edge Detection Parallel Implementation for FPGAs

Christos Gentsos, Calliope-Louisa Sotiropoulou and Spiridon Nikolaidis
Department of Physics
Aristotle University of Thessaloniki
Thessaloniki, Greece
Email: {cgentsos; lsoti; snikolaid}@physics.auth.gr

Nikolaos Vassiliadis
Micro2gen Ltd.
New Technology Park NCSR Demokritos
Ag. Paraskevi, Athens, Greece
Email: nivas@micro2gen.com

Abstract—**Edge detection is one of the most fundamental algorithms in digital image processing. The Canny edge detector is the most implemented edge detection algorithm because of its ability to detect edges even in images that are intensely contaminated by noise. However, this is a time consuming algorithm and therefore its implementations are difficult to reach real time response speeds. Especially nowadays where the demand for high resolution image processing is constantly increasing, the need for fast and efficient edge detector implementations is ever so present. A new parallel Canny edge detector FPGA implementation is proposed in this paper to answer this demand. This design takes advantage of 4-pixel parallel computations to achieve high throughput without increasing the on-chip memory demands. Synthesis and simulation results are presented to prove the design's efficiency and high frames per second rate.**

Keywords-Canny edge detection, FPGA, parallel architecture, real time

I. INTRODUCTION

Modern image processing applications demonstrate an increasing demand for computational power and memory space. This stems from the fact that image and video resolutions have multiplied in the past few years, especially after the introduction of high definition video and high resolution digital cameras. Therefore there is a need for image processing implementations that can perform demanding computations on substantial amounts of data, with high throughput, and often need to meet real-time requirements.

Edge detection is the first step in many computer vision algorithms. It is used to identify sharp discontinuities in an image, such as changes in luminosity or in the intensity due to changes in scene structure. Edge detection has been researched extensively. A lot of edge detector algorithms have been proposed, such as Robert detector, Prewitt detector, Kirsch detector, Gauss-Laplace detector and Canny detector. Among all the above algorithms, Canny algorithm [1] is the most widely used due to its good performance and its ability to extract optimally edges even in images that are contaminated by gaussian noise. Canny algorithm has the ability to achieve a low error-rate by eliminating almost all non-edges and improving the localization of all identified edges.

Because of its algorithmic efficiency and applicability many Canny implementations have been proposed. In [2] an implementation of a self-adapt threshold Canny algorithm is proposed. This design is FPGA based and intended for a mobile robot system. The results presented are for an Altera Cyclone FPGA and the highest frequency achieved is 27MHz, which result in 2.5ms computation time for a 360x280 grayscale image. In [3] an industrial implementation for ceramic tiles defect detection is presented, which defines the hysteresis thresholds with a histogram subtraction method. A Canny edge detection on NVIDIA CUDA is presented in [4], which takes advantage of the CUDA framework to implement the entire Canny algorithm on a GPU. It achieves a 10.92ms computation time for a 1024x1024 image. In [5] there is an implementation of an adaptive edge-detection filter on an FPGA using a combination of hardware and software components proposed by Altera. In [6] a reconfigurable architecture and implementation of edge-detection using Handle-C is presented This is a pipelined design of a canny-like edge detection algorithm. It achieves a computation time of 4.2ms for a 256x256 grayscale image.

Ours is a novel implementation of a Canny edge detector that takes advantage of 4-pixel parallel computation. It is a pipelined architecture that uses on-chip BRAM memories to cache data between the different stages. The exploitation of both hardware parallelism and pipelining creates a very efficient design that has the same memory requirements as a design without parallelism in pixel computation. This results in achieving increased throughputs for high resolution images and a computation time of 3.09ms for a 1.2Mpixel image on a Spartan-6 FPGA. We present synthesis and simulation results for low-end and high-end Xilinx FPGAs and have achieved higher speeds, better throughputs and efficiency than the implementations presented above.

II. CANNY EDGE DETECTION ALGORITHM

The block diagram of the Canny algorithm is demonstrated in Fig. 1. The Canny algorithm first smoothes the image to eliminate noise by using a smoothing filter such as the Gaussian Convolution. The Gaussian smoothing is performed by using a mask (matrix) which is sled over the image, manipulating a square of pixels at a time. The bigger the

This work has been supported by National Funding and the European Regional Development Fund in the frame of ΕΣΠΑ 2007-2013 under grant No MIKPO2-49-project LoC.

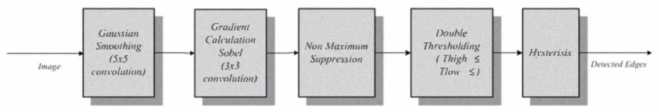
Image
Gaussian Smoothing (5x5 convolution)
Gradient Calculation Sobel (3x3 convolution)
Non Maximum Suppression
Double Thresholding (Thigh ≤ Tlow ≤)
Hysterisis
Detected Edges

Figure 1. Canny Algorithm block diagram

dimensions of the mask are, the lower sensitivity the detector has to noise. A 5x5 mask is a common choice for the size of a Gaussian filter.

After smoothing the next step is the calculation of the gradient of the image. The gradient calculation leads to the detection of the possible edge strength and direction. This is executed by another convolution with a gradient operator. The most commonly used gradient operators are the Prewitt and the Sobel operators. Both operators perform a 2-D spatial gradient measurement on an image. The Sobel edge detector uses a pair of 3x3 convolution masks, one estimating the gradient in the x-direction (columns) and the other estimating the gradient in the y-direction (rows). The Sobel operators are presented in Fig. 2.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 2. 2-D Sobel operators

Gradient magnitude and orientation is defined by the following equations:

$$|G| = \sqrt{G_x^{\,2} + G_y^{\,2}} \tag{1}$$

$$\theta = \arctan\left(G_y / G_x\right) \tag{2}$$

The following computational step is non maximum suppression, which is used to reduce the edge thickness to improve localization. After non maximum suppression the image may still contain some spurious responses. These responses which are called 'streaking' can be eliminated by the use of hysterisis thresholding. In this procedure, two threshold values are set, high threshold Th_h and low threshold Th_l, with which the remaining edge gradient values are compared. Any pixel with a value greater than Th_h is presumed to be a definite edge and any pixel with a value greater than Th_l is considered to be a possible edge. Hysterisis is the procedure where every possible edge is eliminated, unless there is a path from this pixel to a pixel with a gradient above Th_h which includes only pixels with values above Th_l.

III. HARDWARE IMPLEMENTATION

The final goal of our implementation is to use the edge detection stage as a precursor step to feature extraction. This design will be part of a demanding machine vision system, therefore there is a substantial need for efficiency and power in our implementation with as limited use of resources as possible. The input images are 8-bit grayscale. The nature of our implementation requires a successful edge detection for source images contaminated by noise, therefore the adoption of an implementation of the Canny Algorithm was the most suitable option. To achieve high clock rates and throughput, a pipelined design was chosen but with an exploitation of parallel pixel computation. Our block design follows the exact procedure of the Canny algorithm, therefore our implementation has 5 blocks:

- Gaussian Smoothing
- Sobel Gradient calculation
- Non Maximum Suppression
- Double Thresholding
- Hysterisis

A. *Gaussian Smoothing*

The first computational block is the Gaussian smoothing. As previously described, in Gaussian smoothing a mask is convolved with the image pixels to produce an image with reduced noise. We found that a 5x5 mask was sufficient for our implementation. Therefore, for the calculation of one pixel the contents of 25 pixels are required. We chose to implement a design that takes advantage of 4-pixel parallel calculation, for which, as demonstrated in Fig. 3, 40 pixels are required. So for an 8-bit pixel and 32-bit word of pixels, the cache reads are minimized by following the pattern demonstrated in Fig. 4.

For the pixel input a specialized cache is designed. Five cache memory elements are used, each one assigned to storing the data of one image line. We exploit the on-chip BRAMs, which in modern FPGAs is both abundant and fast. Therefore, each BRAM has a size of image width / 4 x 32 bits to accommodate a line of data aligned in 4-pixel words. The calculation of the first pixels begins as soon as the first 2 lines (2x image width) of the cache are filled, since for the first line of borderline pixels the non-existing lines necessary for the calculations are considered to be black. Therefore 2 lines of data are already loaded in the cache and the third line is simultaneously loaded in the cache and directed in the calculating core. Simultaneously, we adjust the matrix normalization factor appropriately to maintain uniformity. Thus, the same size of on-chip memory is used as if the calculation for only one pixel was executed at a time. The results are stored in another cache that serves as an input for the following stage.

B. *Sobel Gradient Calculation*

In the Sobel gradient calculation block the same pixel parallelism principles are applied as in the Gaussian smoothing block. For the calculation of one pixel gradient 9 pixels are required, while for the calculation of four 18 are required (Fig. 5). The gradients for both directions are calculated in parallel as well. For caching, we allocate 3 BRAMs for respective lines. Each BRAM has a size of image data / 4 x 32bits in 4-pixel words, as in the previous stage. For the start of the block calculation the existence of only one full BRAM is required. The direction of the gradient is calculated by using fixed point arithmetic and by implementing the multiplication with shifts

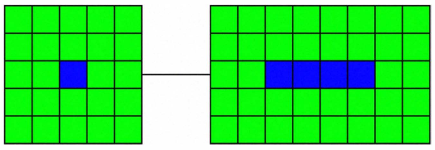

Figure 3. 4-pixel parallel Gaussian smoothing calculation

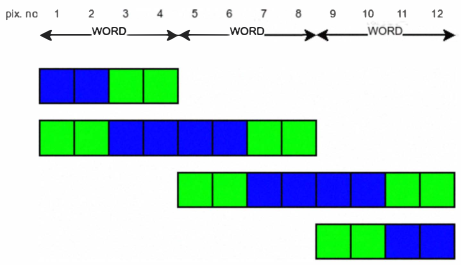

Figure 4. Cache line read pattern

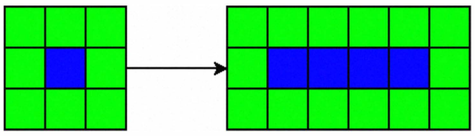

Figure 5. 4-pixel parallel Sobel gradient calculation

and addition/subtraction to increase speed. The results are stored in a cache used as an input for the next stage.

C. Non Maximum Suppression

The Non Maximum Suppression also requires an 8-pixel neighborhood for the determination of each pixel's value. Therefore the parallelism implemented for the 4-pixel simultaneous calculation is the same as in Fig. 5. As in Sobel gradient calculation NMS starts as soon as one cache line is filled with data.

D. Double Thresholding and Hysterisis

Double thresholding is executed by a double comparator. No caching is required before thresholding and the data go right through the next stage of hysteresis. The data produced by the double thresholding stage has a size of 2bits per pixel, as three different values need to be stored for each pixel, no edge, definite edge and possible edge.

The execution of Hysterisis stage also starts as soon as the first pixel data is received. At the same time the data is stored in a specialized cache for this stage with a size of 1x image width x 2bits. Hysterisis is basically a procedure of comparing the value of the pixel at hand with the values of three neighboring pixels above it and the one directly on the left. If the pixel is a possible edge and one of the aforementioned neighboring pixels is a definite edge, then the pixel becomes a definite edge. Otherwise it is left as is. The original algorithm requests a comparison between all 8 neighboring pixels, but testing of our implementation has demonstrated that this comparison is adequate as long as a second pass is executed with the pixels read in the opposite direction. The comparison is also executed for 4 pixels in parallel. The data produced from the first pass of hysterisis is still 2 bits for each pixel and it is stored in an external onboard memory (suitable for the size of the frame we use). For the second pass the image data is read in reverse, from the low right corner to the top left. The hysterisis is executed in exactly the same manner, and finally all the remaining possible edges are suppressed. The final output of the process is only 1bit for each pixel and with all the definite edges detected.

IV. Experimental Results

The above Canny implementation was synthesized for three different Xilinx FPGAs ranging from the older and more economical Spartan-3E to a top-end Virtex 5 FPGA by using the Xilinx ISE 12.1 toolchain [7]. The results are presented in Table I. As demonstrated, in the newer Spartan 6 (45nm technology) and Virtex 5 (65nm technology) FPGAs the final design implementation occupies only a small percentage of the FPGA's total area, and at the same time it achieves a high operating frequency of above 200MHz. Even in the small Spartan-3E (90nm technology) FPGA our design implementation achieves an operating frequency of 120MHz and still leaves more than 70% of the FPGA free for use. The total on-chip memory used is 24kbytes for all types of FPGA.

TABLE I. Canny Synthesis Results

Synthesis Results	Gauss	Sobel	NMS	Db_Thres	Hysterisis	Total	Total(%)	Frequency (MHz)
Spartan 3E Slices	2613	1054	649	37	84	4200	28%	120.4
Spartan 6 Slices	2418	1391	651	36	126	4560	2%	201.4
Virtex 5 Slices	2409	1389	648	40	124	4553	6%	292.8

Our Canny implementation will be used by a lab on chip system on a Spartan 6 FPGA. Therefore we use the results produced by the Spartan 6 synthesis for our simulation. We simulate the design by using 3 different image file sources which are 8bit grayscale files of varying sizes. In Fig. 6 the input and the output files of the Canny implementations are demonstrated. Fig. 6.c and 6.d is an example frame of a video for a lab on chip experiment. The timing results are presented in Table II.

TABLE II. Canny Time Results

Image File	Size	Time (ms)
lena	512x512	0.66
HCLAChip	960x540	1.31
Disc-brake	1280x960	3.09

As can be seen for an image of 1.2Mpixel we have achieved a computation time of 3.09ms. Thus, for an image of 1Mpixel a rate of 396 frames per second has been achieved, which is far beyond our set specifications for a real-time design. On the Virtex-5 the throughput reaches the number of 580 frames per second for 1Mpixel images. Even in the low-end Spartan-3E

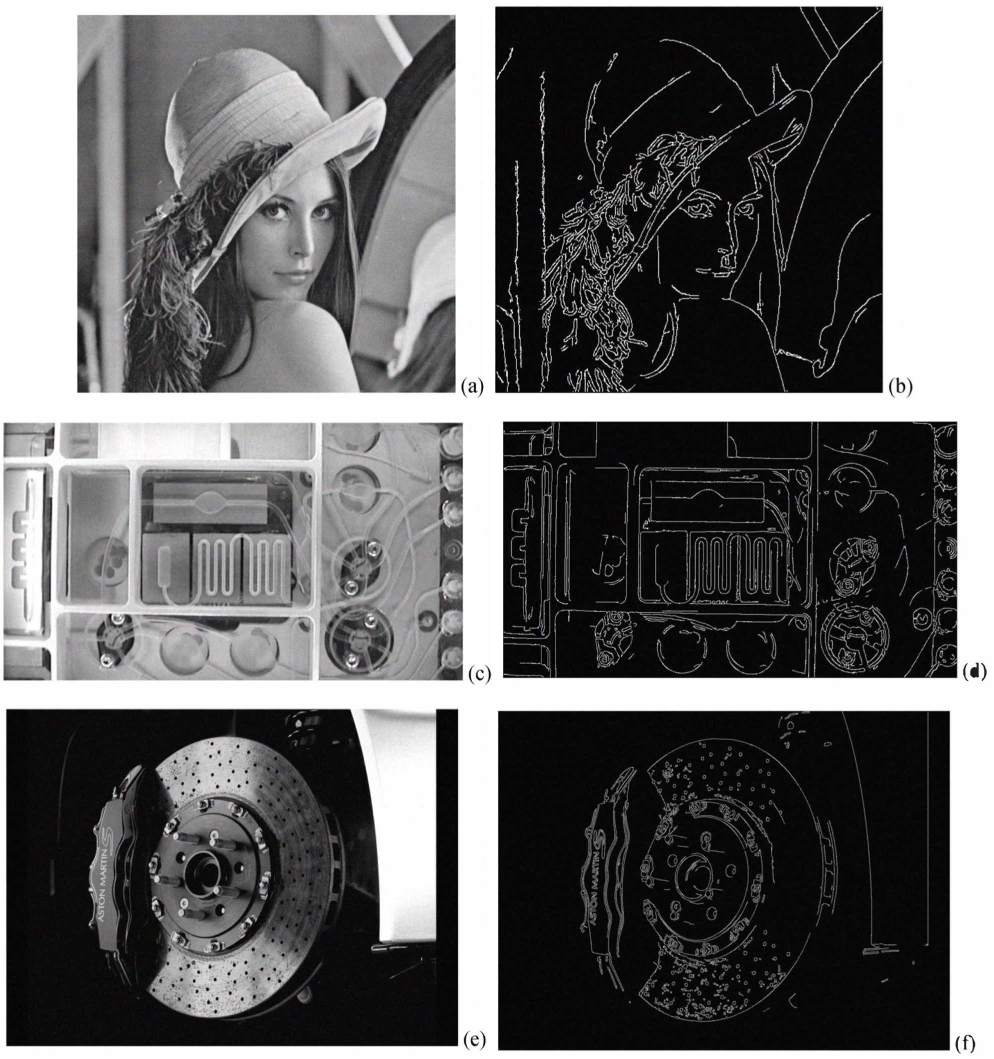

Figure 6. (a) lena input, (b) lena output, (c) HCLAchip input, (d) HCLAchip output, (e) disc-brake input, (f) disc-brake output

FPGA a throughput of 240 frames per second has been reached for the same image size.

V. Conclusions

In this paper a parallel design of a real-time Canny implementation is presented. In this Canny edge detector a parallel architecture of simultaneous 4-pixel calculation is proposed, which increases the throughput of the design without increasing the need for on-chip cache memories. This design has been synthesized for low-end and high-end Xilinx FPGAs, achieving a rate of 240 frames per second for 1Mpixel images on a Spartan-3E occupying a 28% of the area of the chip, to a rate of 580 frames per second on a Virtex-5 occupying a 6% of the area of the chip. In Spartan 6 a computation time of 3.09ms was achieved for a 1.2Mpixel 8-bit grayscale image and a rate of 396 frames per second for 1Mpixel images, while only 2% of the total area of the chip is occupied.

References

[1] J.F. Canny, "A computation approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no 6, pp. 769-798, November 1986

[2] W. He and K. Yuan, "An improved Canny Edge Detector and its Realization on FPGA," Proc. of 7th World Conference on Intelligent Control and Automation, 2008

[3] H. Zeljko, V. Suzana and H. Verica, "Improved Canny Edge Detector in Ceramic Tiles Defect Detection," IEEE Industrial Electronics, IECON 2006 – 32nd Annual Conference, pp. 3328-3331, November 2006

[4] Y. Luo and R. Duraiswami, "Canny Edge Detection on NVIDIA CUDA," Proc. Of IEEE Computer Vision and Pattern Recognition Workshops, 2008 , pp. 1-8

[5] H.S. Neoh and A. Hazanchuk, "Adaptive Edge Detection for Real-Time Video Processing using FPGAs," Altera Corp.

[6] D. V. Rao and M. Venkatesan, "An Efficient Reconfigurable Architecture and Implementation of Edge Detection Algorithm Using Handle-C," Proc of International Conference on Information Technology: Coding and Computing, ITCC 2004, Vol. 2, pp. 843-847

[7] "Xilinx ISE 12.1 – Synthesis and Simulation Design Guide", Xilinx Corp.