VOL. 10, NO. 7, APRIL 2015 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences


www.arpnjournals.com

ALGORITHM AND IMPLEMENTATION OF DISTRIBUTED CANNY EDGE DETECTOR ON FPGA

Aravindh G. and Manikandababu C. S.
Department of ECE Sri Ramakrishna Engineering College Coimbatore, India
E-Mail: aravindhvlsi@gmail.com

ABSTRACT

Edge detection is one of the most commonly used operations in image analysis particularly in the areas of feature extraction. Edge in an image indicates the boundaries between overlapping objects. An edge represents the boundary between an object and the image background, hence if the edges are identified with high accuracy in an image then all its objects can be located and basic properties of an image can also be measured. An edge can also be defined as a set of connected pixels that forms a boundary between two disjoints regions. Edge detection is a basic method of segmenting an image into regions of discontinuity. The data which are extracted in edge detection is too large, so to achieve the high speed of image processing is a difficult task. To solve this problem, a distributed canny edge detection algorithm is proposed that results in significant reduction of memory requirements with decreased latency and increased throughput with no loss in edge detection performance as compared to the original canny algorithm. In addition, the new algorithm uses a non uniform gradient magnitude histogram to compute block-based hysteresis thresholds. The resultant block-based algorithm has significant reduction in latency and can be easily integrated with other block-based image codecs then it is made capable of supporting fast edge detection of images and videos with high resolution rate, including full-HD videos as the latency is changed as a function of the block size instead of the frame size. In addition to that, quantitative conformance evaluations and subjective tests show that the edge detection performance of the proposed algorithm is better than the original frame-based algorithm, especially for noisy images. Furthermore, FPGA-based hardware architecture of our proposed algorithm is presented in this paper and the architecture is synthesized on the Xilinx Virtex-5 FPGA. Simulation results are dispensed to illustrate the performance of the proposed distributed Canny edge detector. The FPGA simulation results displays that we can process a 512×512 image in 0.287ms at a clock rate of 100 MHz.

Keywords: canny edge detector, distributed image processing, high throughput, parallel processing, FPGA.

1. INTRODUCTION

The edge detection process adopts the simplified analysis of images by drastically reducing the amount of data to be processed, while at the same time it will preserve the useful structural information about object boundaries and features. There is certainly a big deal related to diversity in the applications of edge detection, as there are many applications share a common set of requirements [1]. The Canny edge detector is used in many real-time applications due to its ability to extract significant edges with good detection and good localized features. But unfortunately, the canny edge detection algorithm contains extensive pre-processing and post-processing steps so it is computationally complex than other edge detection algorithms. In a recursively implementable edge detection algorithm is suggested and optimized using retiming techniques and its performance also quite poor in images with low SNRs. The approach projected in [2] combines both the derivative and smoothening operations of the Canny algorithm into a single mask in order to reduce computations. The pipelined implementation is a kind of block-based approach in which block-size is 2 rows of pixels. However it overcomes the dependencies between the blocks by fixing high and low thresholds to a constant value. In the above approaches gradient thresholds are not adapted to the image characteristics; hence their performance is not guaranteed for blurred images and images with low SNRs [2]. Canny edge detector a parallel architecture [3] of simultaneous 4-pixel evaluation is proposed, which in turn increases the throughput of the design without increasing hr further need for on-chip cache memories. This design is further synthesized for low-end and high-end Xilinx FPGA. However, in [2], the hysteresis thresholds calculation is based on a very finely and uniformly quantized 64-bin gradient magnitude histogram, which is computationally cost effective and further make it difficult for real-time implementation. The CannyDeriche filter [4] is a network with four transputers that detect edges in a 256 × 256 image in 6s, far from the requirement for real-time applications. Although the design in [5] improved the Canny-Deriche filter implementation of [4] and was able to process 25 frames/s at 33 MHz, the used off-chip SRAM memories consist of Last-In First-Out (LIFO) stacks, which in turn increases the area overhead compared to [4]. Demigny pro- posed a new organization of the Canny Deriche filter in [6], which reduces the memory size and the computation cost by a factor of two. However, the number of clock cycles per pixel of the implementation [6] varies with the size of the processed image, resulting in variable clockcycles/pixel from one image size to another with increasing processing time as the image size increases.

The proposed work is focusing on reducing the latency and increasing the throughput of the Canny edge detection algorithm so that it can be used in real-time processing applications. As a first step, the image can be partitioned into blocks and the Canny algorithm can be

applied to each of the blocks in parallel. Each block can be processed simultaneously it reduces the latency significantly. Further it allows the block-based Canny edge detector to be pipelined very easily with existing block-based codecs, thereby improving the timing performance of image/video processing systems. Most importantly, conducted conformance evaluations and subjective tests evaluates that, the proposed algorithm reveals better edge detection results for both clean and noisy images when compared to frame-based Canny edge detector. The block-based Canny edge detection algorithm is mapped onto an FPGA-based hardware architecture. The architecture is flexible enough to handle different image sizes, block sizes and gradient mask sizes. It consists of 32 computing engines configured into 8 groups with 4 engines per group. All 32 computing engines work in parallel lending to a 32-fold decrease in running time without any change in performance when compared with the frame-based algorithm. The architecture has been synthesized on the Xilinx Virtex-5.

In this paper, FPGA synthesis results, including the resource utilization, execution time, and comparison with existing FPGA implementations are presented. The other contents of the paper are organized as follows. Section 2 reveals a brief overview of the related work. Section 3 presents the proposed distributed Canny edge detection algorithm which includes the adaptive threshold selection algorithm and a non-uniform quantization method to compute the gradient magnitude histogram. Quantitative conformances as well as subjective testing results are presented in Section 4 in order to illustrate the edge detection performance of the proposed distributed Canny algorithm as compared to the original Canny algorithm for clean as well as noisy images. In addition, the effects of the gradient mask size and the block size on the performance of the proposed distributed Canny edge detection scheme are discussed and illustrated in Section 4. The proposed hardware architecture and the FPGA implementation of the proposed algorithm are described in Section 5. The FPGA synthesis results and comparisons with other implementations are presented in Section 6. Finally, conclusions and results are presented in Section 7.

2. RELATED WORK

R. Ponneela Vignesh *et al.* in [7] implemented the Canny edge detection algorithm in FPGA device, and it is applicable for image segmentation, image tracking, image coding etc. In this paper Canny edge algorithm reduces memory requirements, decreased latency, increased through output with no loss in edge detection performance and Canny edge detection algorithm use probability for finding error rate localization and response in various images. Chandrashekar N.S *et al.* in [8] explains the distributed Canny edge detection algorithm that results in significant reduction in memory requirements with decreased latency and increased throughput with no loss in edge detection performance as compared to the original Canny algorithm. Tejaswini H.R *et al.* in [9] explains that the edge detection is an eloquent step in image processing and in object recognition. The Canny edge detection is called as optimal detection due to its good performance. Samina Jafar *et al* in [10] explains the edge detection is one of the key stages in image processing and objects reorganization. The Canny Edge Detection is one of the most widely used edge detection algorithm due to its good performance.

T. Rupalatha *et al* in [11] explains edge detection is one of the basic operation carried out in image processing and object identification. In this paper, the distributed Canny edge detection algorithm that results in significant reduction in memory consumption with no loss in edge detection performance as compared to the original Canny algorithm. In [12] a long standing challenge in the field of image processing is that intensive computation power is required to achieve high accuracy and real-time performance. Recently, GPU has evolved into an extremely powerful computation resource. For example, NVIDIA GTX 280 with 240 processing cores at 602MHzand 1GB of GDDR3running through a 512-bit memory bus performs 933 GFLOPS in its peak performance. As a comparison, 3.2 GHz Intel Core2 Extreme (QX9775) operates at roughly 51.2 GFLOPS. The selected algorithms are parallelized efficiently on the GPU. A set of metrics was proposed to parameterize quantitatively the characteristics of parallel implementation of selected algorithms.

These results can be shared and employed by other researchers to predict the appropriateness of their algorithm for parallel implementation [13]. In [14] Modern image processing applications demonstrate an increasing demand for computational power and memory space. Because of its algorithmic efficiency and applicability many Canny implementations have been proposed. In this 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 continuous usage 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. In this paper a parallel design of a real-time Canny implementation is presented. This design has been achieve a rate of 240 frames per second for 1Mpixel image on sparttan -3E occupying a 28% of the area on chip [15].

3. PROPOSED DISTRIBUTED CANNY EDGE DETECTION ALGORITHM

The superior performance of the frame-based Canny algorithm is due to the fact that it computes the gradient thresholds by analyzing the histogram of the gradients at all the pixel locations of an image. Though it is purely based on the statistical distribution of the gradient values, it produces better results on natural images which consist of three regions namely smooth, texture regions and high-detailed regions [16]. Directly applying the frame-based Canny at a block-level would fail because such a mix of regions may not be available

locally in every block of the frame. This would lead to excessive edges in texture regions and loss of significant edges in high detailed regions. Figure-1 represents the block diagram of the canny edge detection algorithm.

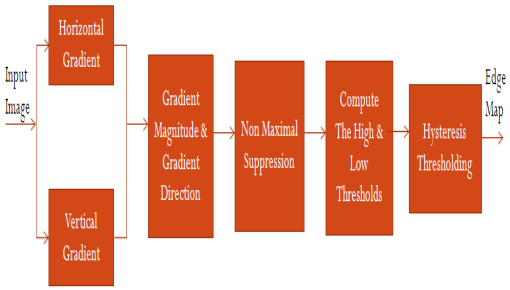


Figure-1. Block diagram of the existing Canny edge detection.

The Canny edge detection algorithm operates on the whole image and has a latency that is proportional to the size of the image mean while performing the original Canny edge detection algorithm at the block level would speed up the operations, it would result in loss of important edges in high-detailed regions and excessive edges in texture regions. Basically natural images consist of a mix of three regions namely smooth, texture and high-detailed regions and such a mix of regions may not be available locally in every block of the entire image. The Canny edge detection algorithm operates on the whole image and has a latency that is proportional to the size of the image. While performing the existing Canny edge algorithm at the block-level would speed up the operations, but there are significant loss of edges in high-detailed regions and excessive edges in texture regions. Natural images comprise of three regions namely smooth, texture and high-detailed regions and such a mix of regions may not be available locally in every block of the entire image. In [17], we proposed a distributed Canny edge detection algorithm, which takes off the dependency between the various blocks so that the image can be divided into blocks and each block can be processed in parallel. The input image is divided into $m \times m$ overlapping blocks. The adjacent blocks overlap by $\frac{L-1}{2}$ pixels for a $L \times L$ gradient mask. However, for each of the block, only edges in the central n× n (where n = m + L − 1) non-overlapping region are included in the final edge map. Steps followed in 1 to 4 and in Step 6 of the distributed Canny algorithm are the same as in the original Canny algorithm except that these are now applied at the block level. The step 5, is the calculation of hysteresis, high and low thresholds calculation, is modified to enable parallel processing. In [17], a parallel hysteresis thresholding algorithm was proposed based on the observation that a pixel with a gradient magnitudes that are calculated in 2, 4 and 6 corresponds to blurred edges in an image, deliriously visual edges and very sharp significant edges, respectively. In order to evaluate the high and low hysteresis thresholds, we compute very finely and uniformly quantized 64-bin gradient magnitude histograms over overlapped blocks. The 64-bin uniform discrete histogram is used for the high threshold calculation, this entails performing 64 multiplications and $64 \times Np$ comparisons.

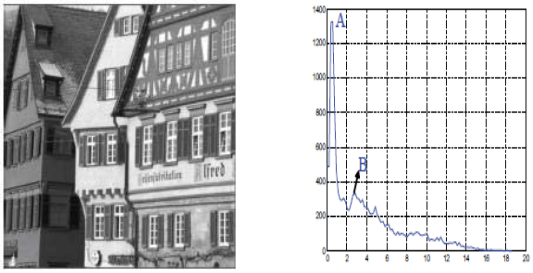


Figure-2(a). Original 512×512 House image; (b) Histogram of the gradient magnitude after non-maximal suppression of the House image.

As in [18], it was observed that the largest peak in the gradient magnitude histograms after NMS of the Gaussian smoothed natural images occurs near the origin and corresponds to low-frequency content, while edge pixels form a continuous series of smaller peaks where each peak corresponds to a class of edges having similar gradient magnitudes. Subsequently, the high threshold values should be selected between the largest peak and the second largest edge peak.

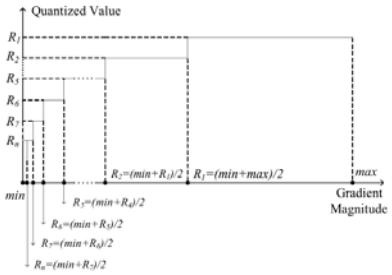


Figure-3. Reconstruction values and quantization levels; min and max representation.

A sample gradient magnitude histogram is represented in Figure-2(b) for the 512×512 House image Figure-2(a). Based on the above observation, we design a non-uniform quantizer to discretize the gradient magnitude histogram. Peculiarly, the quantizer needs to have more quantization levels in the region between the largest peak A and the second largest peak B and few quantization levels in other parts. Figure-3 reveals the schematic diagram of the designed quantizer. As a result, n reconstruction levels can be computed as follows: $R_1 = \frac{min + max}{n}$, $R_{i+1} = \frac{min + R_i}{n}, i = 2,\ldots,n$ where min and max values represent, the minimum and maximum values of the gradient magnitude after NMS, and R_i is the

reconstruction level. The proposed distributed Canny edge thresholds selection algorithm is represented in Fig. 4. Let G_t be the set of pixels with gradient magnitudes greater than a threshold t, and let NG_t for t = 2, 4, 6, be the number of corresponding gradient elements in the set G_t. Using NG_t, an intermediate classification threshold C is calculated to indicate whether the considered block is high-detailed region, moderately edged region, blurred or textured region, as represented in Figure-4.

Consequently, the set $G_t = G_c$ can be selected for computing the high and low thresholds. The calculation of high threshold is based on the histogram of the set G_c such that 20% of the total pixels of the block would be identified as strong edges. The lower threshold is the 40% percentage of the higher threshold as in the original Canny algorithm. The comparison is made for the high threshold value that is calculated using the proposed distributed algorithm based on an 8-bin non-uniform gradient magnitude histogram with the value obtained when using a 16-bin non-uniform gradient magnitude histogram. The above two high thresholds values are similar. Therefore, we can make use of the 8-bin non-uniform gradient magnitude histogram in our implementation. The pseudo-code of the block classification technique and the proposed adaptive threshold selection algorithm is shown as follows.

Pseudo-code for the proposed block classification

Step 1: Pixel Classification

$$pixel\ type = \begin{cases} uniform & var(x,y) \le T_u \\ texture & T_u < var(x,y) \le T_e \\ edge & T_e < var(x,y) \end{cases}$$

Step 2: Block Classification

Block Type	No. of pixel of pixel type	
	$N_{uniform}$	N_{edge}
Smooth	≥0.3(Total_Pixel)	0
Texture	<0.3(Total_Pixel)	0
Edge/ Texture	<0.65(Total_Pixel-N_{edge})	(>0)&(<0.3(Total_Pixel))
Medium Edge	≥0.65(Total_Pixel-N_{edge})	(>0)&(<0.3(Total_Pixel))
Strong Edge	≤0.75(Total_Pixel)	≥0.3(Total_Pixel)

$var(x,y)$: the local (3 × 3) variance at pixel (x,y);
T_u and T_e: Two thresholds as in [7];
Total_Pixel: The total number of pixels in the block;
$N_{uniform}$: The total number of uniform pixels in the block;
N_{edge}: The total number of edge pixels in the block;

Pseudo-code for the Proposed Adaptive Threshold Selection Scheme

Let P_l be the percentage of pixels, in a block, that would be classified as strong edges.
Step 1: If smooth block type
$P_l = 0$; /*No edges*/
else if texture block type
$P_l = 0.03$; /*Few Edges*/
else if medium edge block type
$P_l = 0.02$; /* Medium Edges */
Else
$P_l = 0.2$; /*Meany Edges*/
Step 2: Compute the 8-bin non-uniform gradient magnitude histogram and corresponding cumulative distribution function F (G).
Step 3: Compute High_thershold as F(High_thershold)=1-P_l
Step 4: Compute Low_thershold=0.4*High_thershold

4. PROPOSED DISTRIBUTED CANNY ALGORITHM IMPLEMENTATION ON FPGA

In this section, we describe the hardware implementation of our proposed distributed Canny edge detection algorithm on the Xilinx Spartan-3E FPGA.

a) Architecture overview

Depending on the available FPGA resources, the image has to be sub-divided into q sub-images and each sub-image is further divided into p m x m blocks. The proposed architecture, represented in Figure-4, inheres of q processing units in the FPGA and some Static RAMs (SRAM) organized into q memory banks to store the image data, where q corresponds to the image size divided by the SRAM size. Each processing unit processes a sub-image and reads/writes data from/to the SRAM through ping -pong buffers that are implemented on the FPGA with the use of dual port Block RAMs (BRAM). As represented in Figure-4, each processing unit (PU) consists of p computing engines (CE), where each CE detects the edge map of an $m \times m$ block image. Thus, $p \times q$ blocks can be processed at the same time and the processing time for an $N \times N$ image is reduced, in the best case, by a factor of p x q.

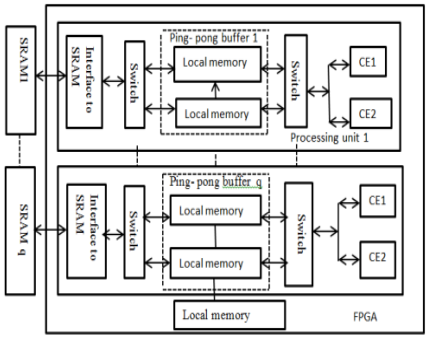


Figure-4. The architecture of the proposed distributed Canny algorithm.

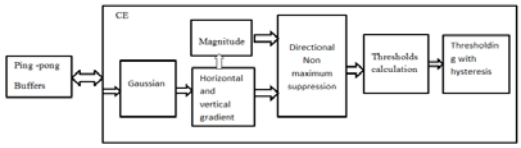
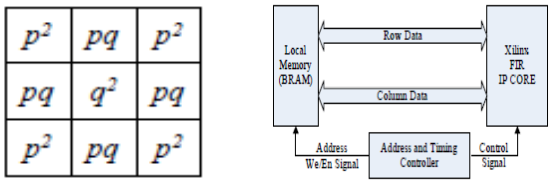


Figure-5. Block diagram of the CE (compute engine) for the proposed distributed Canny edge detection.

The specific values of p and q depend on the processing time of each PE, the loading time of the data from the SRAM to the local memory and the interface between FPGA and SRAM, similar to the total pins on the FPGA, the width of the data bus, the width of the address bus and the maximum system clock of the SRAM. In our processing application, we choose p = 2 and q = 8. In the proposed distributed Canny edge detection architecture, each CE consists of the following units, as represented in Figure-5.

b) Image smoothening

The input image is smoothened using a 3×3 Gaussian mask, as represented in Figure-6(a). The Gaussian filter Figure-6(a) is separable and, thus, the implementation of the 2-D convolution with the 3×3 Gaussian mask is achieved using row and column 1- D convolutions. The put forwarded architecture for the smoothening unit is represented in Figure-6(b).



(a) (b)

Figure-6. (a) Mask for the low pass Gaussian filter with p = 0.0437; q = 0.9947; (b) Pipelined image smoothening unit.

The main components of the architecture consist of a 1-D finite impulse filter (FIR) to process the data and the on-chip Block RAM (BRAM) to store the data. In our proposed design, we take up the Xilinx's pipelined FIR IP core, which dispenses a highly parameterizable, efficient in area, high-performance FIR filter utilizing the structure characteristics in the coefficient set, such as symmetry and conjugacy .By exploiting the symmetry of the Gaussian filter, the architecture make use of two multipliers to perform the 1-D convolution using a 3-tap filter. The address controller bring in the input image data from the local memory into the FIR core and, after the computation, it stores the results back in the BRAM.

c) Gradients and gradient magnitude calculation of an image

This stage calculates the vertical and horizontal gradients using convolution kernels. The kernels vary in size from 3×3 to 9×9, depending on the sharpness of the image. The Xilinx FIR IP core, which is able to support up to 256 sets of coefficients with 2 to 1024 coefficients per set, is further used to implement the kernels. The entire design is pipelined, and the corresponding output is generated for every clock cycle. This output is made as an input to the magnitude calculation unit which computes the values at each location of pixels and the gradient magnitude from the pixel's horizontal and vertical gradients.

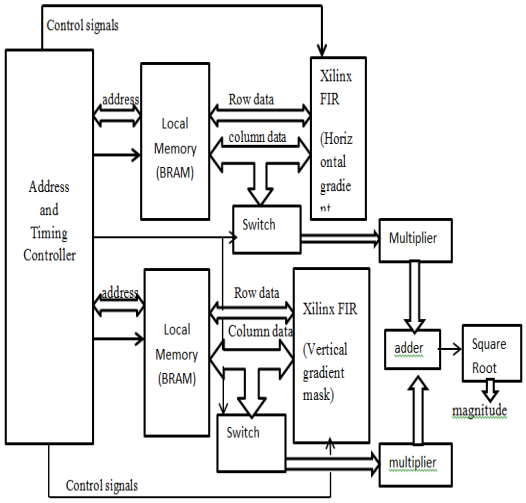


Figure-7. Gradient and magnitude calculation unit.

The architecture of this unit is represented in Figure-7. The gradient calculation architecture of an image contains two 1-D FIR models and the corresponding local memory. The filters that are used for computing the horizontal and vertical gradient elements can process data in parallel. The gradient magnitude computation comprises of two multipliers and one square- root computation module that are implemented by the Xilinx Math Function IP cores.

d) Directional non maximum suppression

Figure-8 shows the architecture of the directional non-maximum suppression unit.

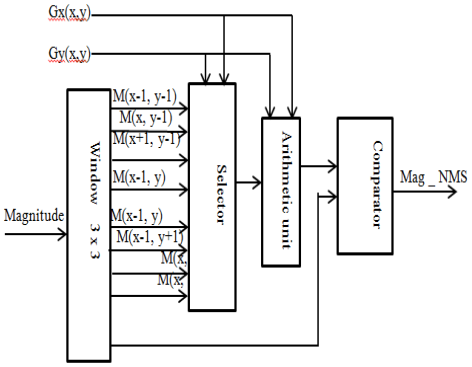


Figure-8. Directional non maximum suppression unit.

In order to make access all the pixels' gradient magnitudes in the 3×3 window at the same time, two First Input First Output buffers (FIFO) are employed.

The horizontal gradient G_x and the vertical gradient G_y control the selector which delivers the gradient magnitude (marked as M(x, y) in Figure-8) of nearest values along the direction of the gradient, into an arithmetic unit.

e) Calculation of the hysteresis threshold values

Since the low and high thresholds are calculated based on the gradient histogram values, from that we need to compute the histogram of the image after it has undergone directional non-maximum suppression. As described in Section 3, proceeding 8-step non-uniform quantizer is employed to obtain the discrete histogram for each processed block.

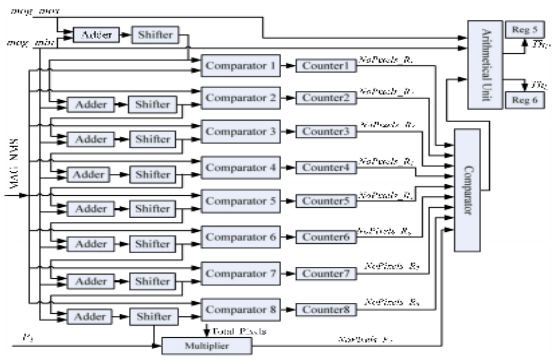


Figure-9. The architecture of the Threshold calculation unit.

f) Thresholding with hysteresis

Since the output of the non-maximum suppression unit contains some spurious edges, the method of finding threshold values with hysteresis is used.

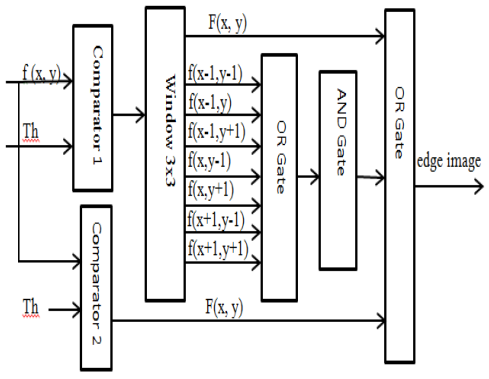


Figure-10. Pipelined architecture of the Thresholding unit.

Two threshold values, high threshold ThH and low threshold ThL, which are obtained as a result from the threshold calculation unit, are employed. Let $f_{ab}(x, y)$ be the image obtained from the non maximum suppression stage, $f_{11}(x, y)$ be the strong edge image and $f_{12}(x, y)$ be the weak edge image.

5. MATLAB EXPERIMENTAL RESULTS

a) Parametrical analysis

The performance of the proposed algorithm is affected by two major parameters, which includes the mask size and the block size. the size of the gradient mask is a function of the standard 1) The Effect of Mask Size: deviation σ of the Gaussian filter, and the best choice of σ is based on the image characteristics. Canny has shown in [19] that the optimal operator for detecting step edges in the presence of noise is the first derivative of the Gaussian operator. As stated in Section 2, for the original Canny algorithm as well as the proposed algorithm, this standard deviation is a parameter that is typically set by the user based on the knowledge of sensor noise characteristics. It can also be set by a separate application that estimates the noise and/orblur in the image. A large value of σ results in smoothing and improves the edge detector's resilience to noise, but it undermines the detector's ability to detect the location of true edges.

In contrast, a smaller mask size (corresponding to a lower σ) is better for detecting detailed textures and fine edges but it decreases the edge detector's resilience to noise. An L-point even-symmetric FIR Gaussian pulse-shaping filter design can be obtained by truncating a sampled version of the continuous-domain Gaussian filter of standard deviation σ. The size L of the FIR Gaussian filter depends on the standard deviation σ and can be determined as follows: where CT represents the cut-off value in the spatial domain of the continuous-domain Gaussian function and determines the cut-off error. Calculation of Block Size: To find out the smallest block size for which the proposed Canny algorithm can detect all the psychovisually important edges, the sensed visual quality of the obtained edge maps was assessed using visual quality metrics.

b) Edge detection performance analysis

The edge detection result performance of the proposed split approach is analyzed by comparing the perceptual significance of its resulting edge map with the one produced by the original.

(a) (b) (c)

Figure-11. Comparison of the edge maps of noisy images by using the originalCanny edge detector and the proposed method: (a) images with Gaussian white noise (σn = 0.01); edge-maps of (b) the original Canny edge detector, and (c) the proposed algorithm with a non-overlapping block size of 64 × 64, using a 9 × 9 gradient mask to noise than the original frame-based Canny.

To further assess the performance of the proposed Split Canny algorithm, similar quantitative evaluations and subjective tests are performed. The conformance evaluations aim to evaluate the similarity between edges detected by the original frame-based Canny algorithm and the proposed distributed Canny edge detection algorithm, while the subjective tests aim to validate whether the edge detection performance of the proposed distributed Canny is better, worse, or similar to the original frame-based Canny as perceived by subjects. 1) Conformance Evaluation: In order to quantify the similarity of two edge maps, three metrics, Pco (percentage of edge pixels detected by both implementations) Pnd (percentage of edge pixels detected by the original Canny edge detection2) Subjective Testing: 2) Subjective tests: were conducted by having human subjects evaluate the quality of the detected edge maps that are generated by the proposed algorithm and the original Canny for both clean and noisy images, without the subjects knowing which algorithm produced which edge maps, using images from the SIPI Database [20] and the Standard Test Image Database [21].

c) Synthesis results

The proposed FPGA-based architecture can support multiple image sizes and block sizes. To demonstrate the performance of the proposed system, a Xilinx Virtex-5 FPGA was used to process grayscale images with a block size of 64 × 64. The data width is 16 bits (Q8.7) with 8 bits to represent the integer part since the maximum gray value of the image data is 255, and 7 bits to represent the fractional part since the Gaussian filter parameters are decimals. Our analysis shows that 7 bits are sufficient to meet the accuracy requirement of the Gaussian filter parameters, which is typically in the order of 0.001. To store grayscale images, we used the SRAM (CY7C0832BV).

Table-1. Resource utilization on Xc5vsx240t for 1CE.

Block size	Number of CE	Occupied slices	Slice Reg.	Slice LUTs	DSP48Es	Total used memory (KB)
64×64	1	747 (2%)	1270 (1%)	2578 (2%)	7 (1%)	217 (1%)

Table-2. Resource utilization on Xc5vsx240t For 1PU.

Block size	Number of CE	Occupied slices	Slice Reg.	Slice LUTs	DSP48Es	Total used memory (KB)
64×64	4	2988 (8%)	5080 (4%)	10312 (8%)	28 (3%)	2023 (10%)

Table-3. Resource utilization on Xc5vsx240t for an 8-PU architecture

Block size	Number of CE	Occupied slices	Slice Reg.	Slice LUTs	DSP48Es	Total used memory (KB)
64×64	32	23904 (64%)	40640 (32%)	82496 (65%)	224 (25%)	16184 (87%)

Table-4. Clock cycles for each unit.

	T_{GRAD}	T_{FLR}	T_{NMS}	T_{TC}	T_{TH}
Clock Cycles	9248	16	20	4630	4634

This is a dual ported SRAM with 110 pins. The Xilinx Virtex-5 FPGA (XC5VSX240T) has 960 I/O pins and so, to satisfy the I/O pin constraint, the maximum number of PUs is 8 (q = 8). The local memory on the FPGA for a block size of 64 × 64, which is needed to support 8 PUs, is equal to 7 pqm2b = 4046 p Kbits, for q = 8, m = 68 (for a 64 × 64 block size and A 3 × 3 gradient mask size), and b = 16. Since the available memory resource on the FPGA is 18,576 Kbits, the p value using the memory constraint is determined to be 4. The p value could have also been constrained by the number of available slices. Since the number of slices for the considered FPGA is very large (37440) and since each CE only utilizes a small slice percentage, the local memory resource in each PU constrains p, the number of CEs in each PU, and not the numbers of slices. Taking all this into consideration, our design has q = 8Pus and each PU has p = 4 CEs. This design is coded in Verilog and synthesized on a Xilinx Virtex-5 device (XC5VSX240T) using the Xilinx's ISE software and verified using Modelsim. According to the 'Place and Route' synthesis report, our implementation can achieve an operating frequency of 250 MHz. But we choose 100 MHz to support a pipelined implementation of SRAM read/write and CE processing.

d) Experimental results

We present the following experiments to evaluate the effectiveness of the new distributed Canny edge detector that is proposed in this paper.

Figure-12. Floating-point Matlab simulation results for the 512×512 House image: (a) Edge map of the original Canny edge detector; (b) Edge map of the algorithm of [4] with a 3×3 gradient mask and a block size of 64; (c) Edge map of our proposed algorithm with a 3×3 gradient mask and a block-size of 64.

e) Evaluation of fixed-point Mat lab and FPGA simulation results

Figure-13. (a) Edge map of Mat lab implementation; (b) Edge map of FPGA implementation.

Figure-13 shows the fixed-point Mat lab implementation software result and the FPGA implementation generated result for the 512×512 House image using the proposed distributed Canny edge detector with block size of 64×64 and a 3×3gradient mask. The FPGA result is obtained using Model Sim. Hence it is evaluation results shows that the hardware implementation of our proposed algorithm can successfully detect significant edges and results in edge maps that are similar to the ones that are obtained using the fixed-point Mat lab simulation. Furthermore, for a 100MHz clock rate, the total time required by the process for running using the FPGA implementation is 0.28ms for a 512×512 image.

6. CONCLUSIONS

The original Canny algorithm relies on frame-level statistics to predict the high and low thresholds and thus has latency proportional to the frame size. In order to reduce the large latency and meet real-time requirements, we presented a novel Split Canny edge detection algorithm which has the ability to compute edges of multiple blocks at the same time. To support this, an adaptive threshold selection method is proposed that predicts the high and low thresholds of the entire image while only processing the pixels of an individual block. This results in three benefits: 1) a significant reduction in the latency; 2) better edge detection performance; 3) the possibility of pipelining the Canny edge detector with other block-based image codecs. In addition, a low complexity non-uniform quantized histogram calculation method is proposed to compute the block hysteresis thresholds. The proposed algorithm is scalable and has very high detection performance. We show that our algorithm can detect all psycho-visually important edges in the image for various block sizes. Finally, the proposed algorithm is mapped onto a Xilinx Virtex-5 FPGA platform and tested using ModelSim. The synthesized results show 64% slice utilization and 87% BRAM memory utilization. The proposed FPGA implementation takes only 0.721ms (including the SRAM read/write time and the computation time) to detect edges of 512 × 512 images in the USC SIPI database when clocked at 100 MHz. Thus the proposed implementation is capable of supporting fast real-time edge detection of images and videos including those with full-HD content.

REFERENCES

[1] J. Canny. 1986. “A computational approach to edge detection,” IEEE Trans. PAMI, vol. 8, no. 6, pp. 679 – 698, November.

[2] Xu, Qian, Chaitali Chakrabarti and Lina J. Karam. 2011. "A distributed Canny edge detector and its implementation on FPGA." In Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011 IEEE, pp. 500-505. IEEE.

[3] Gentsos, Christos, C-L. Sotiropoulou, Spiridon Nikolaidis, and Nikolaos Vassiliadis. 2010. "Real-time canny edge detection parallel implementation for FPGAs." In Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on, pp. 499-502. IEEE.

[4] R. Deriche. 1987. “Using canny criteria to derive a recursively implemented optimal edge detector,” Int. J. Comput. Vis., vol. 1, no. 2, pp. 167–187.

[5] L. Torres, M. Robert, E. Bourennane and M. Paindavoine. 1995. “Implementation of a recursive real time edge detector using retiming technique,” in Proc. Asia South Pacific IFIP Int. Conf. Very Large Scale Integr. pp. 811–816.

[6] F. G. Lorca, L. Kessal and D. Demigny. 1997. “Efficient ASIC and FPGA implementation of IIR filters for real time edge detection,” in Proc. IEEE ICIP, vol. 2. October. pp. 406–409.

[7] R. Ponneela Vignesh and R. Rajendran. 2012. “Performance and Analysis of Edge Detection Using FPGA Implementation”. nternational Journal of Modern Engineering Research (IJMER) Vol.2, Issue.2, March-April. pp-552-554.

[8] Chandrashekar N.S and Dr. K.R. Nataraj. 2012. "A Distributed Canny Edge Detection and Its Implementation on FPGA" International Journal of Computational Engineering Research (ijceronline.com) Vol. 2 Issue.7. ISSN 2250-3005(online) November.

[9] Tejaswini H.R, Vidhya N, Swathi R Varma and Santhosh B. 2013. "An Implementation of Real Time Optimal Edge Detection and VLSI Architecture". International conference on electronics and communication engineering, 28th April, bengaluru, isbn: 978-93- 83060-04-7.

[10] Samina Jafar and Anupsingh Ramprakashsingh Rajput. 2013. "Improved Distributed Canny Edge Detection In VHDL". VSRD International Journal of Electrical, Electronics & Communication Engineering, Vol. 3 No. 6, June.

[11] T. Rupalatha, Mr. C. Leelamohan and Mrs. M. Sreelakshmi. 2013. "Implementation of Distributed Canny Edge Detection On FPGA". International Journal of Innovative Research in Science, Engineering and Technology, Vol. 2, Issue7, July.

[12] N. D. Narvekar and L. J. Karam. 2011. "A no-reference image blur metric based on the cumulative probability of blur detection (CPBD)," IEEE Trans. Image Process., vol. 20, no. 9, pp. 2678–2683, September.

[13] Gentsos, C. Sotiropoulou, S. Nikolaidis and N. Vassiliadis. 2010. "Real-time canny edge detection parallel implementation for FPGAs," in Proc. IEEE ICECS, December. pp. 499–502.

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

[15] W. He and K. Yuan. 2006. "An improved canny edge detector and its realization on FPGA," in Proc. IEEE 7th WCICA, Jun. 2008, pp. 6561–6564.IEEE Industrial Electronics, IECON 2006 - 32nd Annual Conference , pp. 3328- 3331, November.

[16] S. Varadarajan, C. Chakrabarti, L. J. Karam and J. M.Bauza. 2010. A distributed psycho-visually motivated Canny edge detector,‖ IEEE ICASSP, pp. 822 –825, Mar.

[17] S. Varadarajan, C. Chakrabarti, L. J. Karma and J. M.Bauza. 2010. "A distributed psycho-visually motivated Canny edge detector," IEEE ICASSP, pp. 822 –825, March.

[18] W. He and K. Yuan. 2008. "An improved Canny edge detector and its realization on FPGA," WCICA, pp. 6561 –6564, June.

[19] S. Nercessian. 2009. "A new class of edge detection algorithms with performance measure," M.S. thesis, Dept. Electr. Eng., Tufts Univ., Medford,MA, USA, May.

[20] W. He and K. Yuan. 2008. "An improved Canny edge detector and its Realization on FPGA," WCICA, pp. 6561 –6564, June.