An Optimized FPGA Based Canny Edge Detection System Using High-Level Synthesis

Shahbaz Hassan Khan Malik, shahbazhassankhan.malik.500@my.csun.edu, Shahnam Mirzaei, shahnam.mirzaei@csun.edu, *Member, IEEE*

***Abstract*—The Canny edge detection algorithm is one of the most reliable and popular edge detection algorithms that uses a multi-stage algorithm to detect a wide range of edges in images. This paper presents an optimized hardware accelerated implementation of the Canny edge detection algorithm on Zynq 7000 AP SoC. We offer two optimization algorithms embedded in our Canny edge detection process. Firstly, our implementation uses an optimized approximation algorithm (OAA) to calculate the gradient direction, whilst not compromising the quality of the edge detection process. Our OAA uses significantly less hardware resources compared to alternative CORDIC algorithms used by most of the other implementations. Secondly, we propose a hardware-based optimized hysteresis edge tracking (OHET) algorithm to ensure our IP core produces the final image with consistent and crisp edges. The implementation offered by this paper not only accelerates the edge detection process, but also optimizes the implementation process using high level synthesis (HLS). The proposed implementation also leverages a rich set of directives offered by Vivado HLS tool, further optimizing the design and providing greater control over the hardware implementation. We accelerate the software algorithm by incorporating parallelism into several design components such as noise removal through Gaussian, edge detection using the Sobel operators, non-maximum suppression, edge thinning, and finally hysteresis edge tracking. Our hardware acceleration approach on Xilinx Zynq 7000 AP SoC FPGA running at 100 MHz, achieves a speed up of 2.6-630x in execution time depending on the image size compared to software-based implementation on single CPU Cortex-A9 running at 667 MHz. This offers a significant speed up in performance and computational tasks over its software counterpart. Specifically, our implementation processes a 512×512 image in 11.5 ms at 100 MHz.**

***Keywords*—ZYNQ, FPGA, Reconfigurable Hardware, Image Processing, Hardware Acceleration, Edge Detection, Canny Edge Detection, High Level Synthesis (HLS).**

I. Introduction

Among various edge detection techniques, the Canny edge detection [1] is renowned for its reliability and effectiveness among the many edge detection algorithms that have been created. It offers excellent edge sensitivity together with an innate ability to reduce noise. But for real-time applications on traditional CPU architectures, the Canny algorithm's high computational overhead combined with its iterative and data-dependent nature present formidable obstacles. This computational constraint may be resolved with the help of field programmable gate arrays (FPGAs), which are distinguished by their programmability and parallel processing capabilities. Utilizing FPGA to its fullest potential has historically required a high level of proficiency with hardware description languages (HDL), which adds complexity and lengthens the development period.

The process of developing FPGAs is made much simpler by high level synthesis (HLS) tools [2], which enables developers to specify hardware functionalities in high-level programming languages like C or C++ and then convert those descriptions into HDL code using automated toolsets. This method not only shortens the design cycle but also increases designers' access to FPGA acceleration.

In this paper, we demonstrate a hardware acceleration implementation of Canny edge detection algorithm enabled by HLS on Zynq 7000 AP SoC platform [3]. Our work thoroughly examines a hybrid strategy, demonstrating its superior resource utilization and execution speed compared to conventional software and other hardware-accelerated solutions. In summary, the following is the contribution of our work:

a. To present high performance hardware accelerated Canny edge detection IP core to be integrated into FPGA based image processing platforms.
b. To perform design space exploration through several directives that are provided by Vivado HLS tools.
c. To present two optimization algorithms for the Canny edge detection algorithm that are suitable for hardware implementation: one algorithm approximates the gradient to reduce computational intensity, and the other enhances the hysteresis edge tracking process to connect edges and ensure a final image with continuous and clear edges.

The rest of the paper is organized as follows: Section II presents the related work. Section III describes the design methodology along with two proposed optimization algorithms to improve the resource utilization and edge detection quality. We will review the system architecture implementation and experimental results and analysis in Section IV. Lastly, Section IV concludes the paper with important remarks and future work.

II. Related Work

While there are many implementations of the Canny edge detection across different platforms, only a handful have been specifically designed for reconfigurable hardware. Emrani et al. [4] introduced a parallel approach to accelerate the Canny edge detection on GPU platforms, and showed an improvement of 2-100x in execution time compared to CPU based implementation. The GPU used for this work uses 192 cores at 1.8 GHz system clock rate, and the single CPU uses Intel Core i7 at 3.2 GHz. They have also shown that the execution time grows exponentially with image size. For comparison, we chose a 512×512 image because it is commonly used in the research reviewed in this section. The implementation by [4] has achieved 37.3520 ms for 512×512 image, while processing such image needs 346 ms on single CPU with MATLAB and 59 ms with OpenCV.

Gentsos et al. [5] have implemented the Canny edge detection on FPGA that requires 0.66 ms to process a 512×512 image at 200 MHz. This implementation uses 9106 FFs (no memory usage is reported) as opposed to 3956 FFs proposed by our proposed implementation. More importantly, it suffers

from lack of quality in detecting edges shown by broken edges. The authors introduced an optimized Gaussian smoothing filter in their implementation but did not present additional optimization methods, leading to high FPGA resource utilization. Xu et al. [6] presented an interesting distributed scalable Canny edge detection algorithm that is capable of computing edges of multiple blocks simultaneously. The proposed method detects the edges of 512×512 image within 0.15 ms at 250 MHz clock rate on Virtex-5 FPGA. The implementation consumes significant FPGA area of 47808 FFs and 16,184 KB of memory while it offers a great quality in edge detection process. Our proposed implementation offers comparable edge detection quality with only 8% of resources used by [6]. More importantly, our implementation is not constrained for clock speed and runs at 100 MHz only. Sangeetha et al. [7] presented a block-based edge detection algorithm like [6] except that it offers an approximation method to calculate the direction and magnitude based on CORDIC algorithm by spacing the orientations into different bins. This optimization method reduces the hardware resources to 35,856 FFs, and 278 KB of memory compared to [6] but this is still much higher than proposed algorithm by this paper. The implementation by [8] processes 512×512 image within 0.13 ms at 272 MHz.

The FPGA implementations mentioned in this section [5, 6, 7] focus on the Canny edge detection algorithm and demonstrate relatively good edge detection quality. However, they mostly suffer from the high FPGA area utilization. Our proposed method offers a high-quality detection quality with low FPGA area utilization (3956 FFs, 6267 LUTs, 1504 KB of memory). Our proposed method yields good results and is efficient in terms of FF and LUT resources due to optimization algorithms described in Sections II.B, II.C, and II.D. However, the recursive and random nature of the algorithm presented in II.C consumes high amount of BRAMs. Our implementation offers 11.5 ms execution time for 512×512 image at 100 MHz.

III. Design Approach and Methodology

This section presents the proposed approach for implementing the Canny edge algorithm on Zynq 7000 AP SoC using Zybo development board [8]. Sections A describes the Canny edge algorithm. Sections B, C, and D describe the area and edge detection quality optimization algorithms.

A. Canny Edge Detection Algorithm Development in HLS

The Canny edge detection includes a sequence of steps starting from noise removal through Gaussian filter, edge detection using the Sobel operators, edge magnitude and direction calculations, non-maximum suppression (NMS), double thresholding, and finally hysteresis edge tracking as described by Fig.1 [10]. We have implemented the sequence of steps enumerated in Fig. 1 in C++ as individual functions. Vitis HLS is used to convert high-level software code; in this case C++; into synthesizable RTL. Essentially, each of the steps in the Canny edge detection process is a C++ function of its own which is then sequentially called in a top function. In our design, IPs used in Xilinx FPGAs communicate via a standardized protocol known as AXI (Advanced eXtensible interface) which is specifically designed for high-speed data transfers on Zynq 7000 AP SoC.

The underlying computations involved in edge detection process rely on derivatives; hence, noise is expected. In the Canny edge detection algorithm, this problem is dealt with the application of Gaussian blurring filter to smooth the image. This is done by convolving the image (shown in Fig. 3a) with a Gaussian kernel with result shown in Fig. 3b. The next step of the algorithm applies the Sobel edge detection [9] filter on the blurred image that results in image shown in Fig. 3c. The convolution process is similar to the Gaussian implementation; however, in this case, it produces two values G_X and G_Y which represent the change in intensity in the X and Y directions. The gradient magnitude and direction are then calculated using Eqs. I [11].

$$|G| = \sqrt{(G_x^2 + G_y^2)} \qquad \theta(x,y) = arcta\,n\left(\frac{G_y}{G_x}\right) \qquad \text{(I)}$$

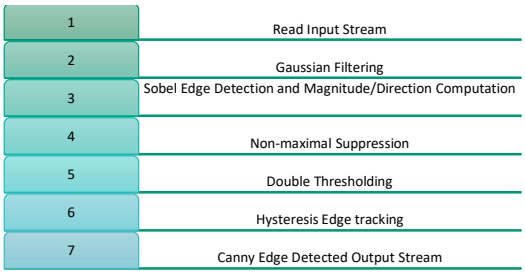

Fig. 1. Sequence of operations used by the Canny edge detection algorithm

Given the limited resources available on the FPGAs, it is very expensive to directly implement an arctangent function in hardware according to Eq. II. Even implementing hardware friendly CORDIC [12] algorithm exceeds the resource utilization. This paper uses an optimized approximation algorithm (OAA) to calculate the gradient direction, which uses significantly less hardware resources while not compromising the quality of the edge detection process.

B. Optimized Approximation Algorithm (OAA)

Specifically, the later stages require the gradient direction to perform a full scan of the image to refine the edges. OAA quantizes the gradient direction into four possibilities by using the relative magnitudes of G_Y and G_X: 0°, 45°, 90°, and 135°. The OAA detects the direction of the gradient by performing comparison of $|G_X|$ and $|G_Y|$. Horizontal and vertical direction can be identified when there is a significant change between $|G_X|$ and $|G_Y|$. Diagonal directions in different directions can be identified according to the signs of G_X and G_Y as shown by the proposed algorithm in Fig. 2.

Optimized Approximation Algorithm (OAA) for Gradient Direction

1: ***Input****: G_X, G_Y*
2: ***Output****: Gradient direction (0: 135° , 1: 0°, 2: 45°, or 3: 90°)*
3:
4: ***Initialization****:*
5: $absG_X \leftarrow |G_X|$; $absG_Y \leftarrow |G_Y|$; $direction \leftarrow -1$
6:
7: ***procedure*** *GradientDirectionDetection (G_X, G_Y)*
8: ***if*** $absG_X > 2 \times absG_Y$ ***then***
9: $direction \leftarrow 1$
10: ***else if*** $absG_Y > 2 \times absG_X$ ***then***
11: $direction \leftarrow 3$
12: ***else***
13: ***if*** $G_X \times G_Y > 0$ ***then***
14: $direction \leftarrow 2$
15: ***else***
16: $direction \leftarrow 0$
17: ***end if***
18: ***end if***
19: ***Output*** *direction*
20: ***end procedure***

Fig. 2. Proposed Optimized Approximation Algorithm (OAA) for gradient detection for the Canny edge detection process

The application of OAA leads to a significant reduction in FPGA resource utilization shown in Table I for the complete the Canny edge detection IP core using HLS tools. Figures 3d and 3e compare the edge detection HLS simulations from the CORDIC and OAA methods proposed in this paper, respectively, demonstrating that our proposed OAA maintains edge detection quality.

TABLE I. COMPARISON OF OAA AND CORDIC RESOURCE UTILIZATION FOR THE CANNY EDGE DETECTION IP CORE USING HLS TOOLS

Angle	BRAM	DSP Blocks	FFs	LUTs
CORDIC	94	24	60,655	86,840
Optimized Approximation Algorithm	94	5	3,956	6,287

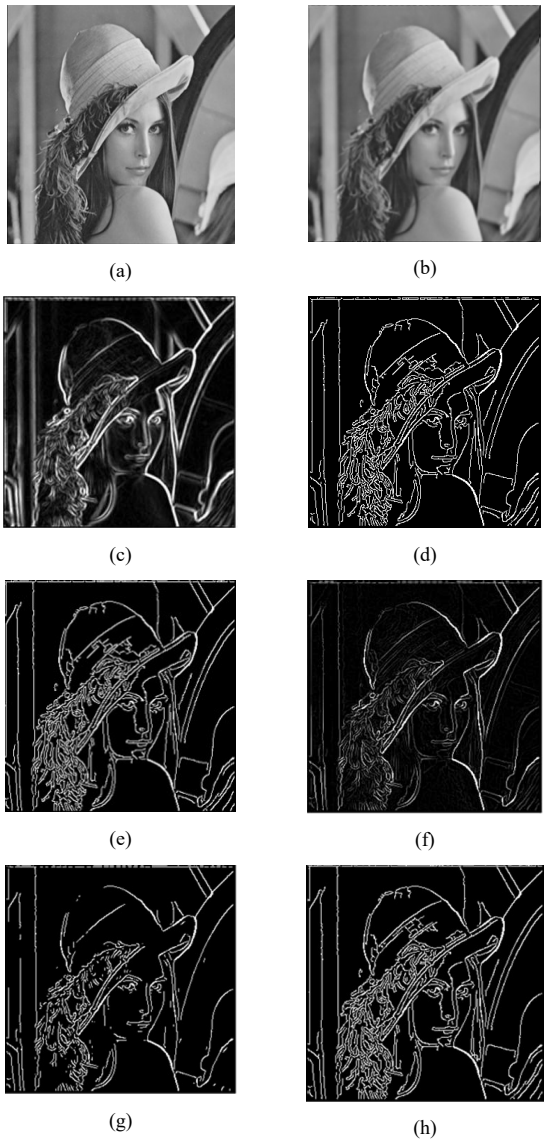

Fig. 3. Edge map result: (a) Original Lena image; (b) Gaussian filter using HLS simulator; (c) Sobel edge detection using HLS simulator; (d) Canny edge detection with CORDIC using HLS simulator; (e) Canny edge detection with OAA using HLS simulator; (f) Canny edge detection with NMS using HLS simulator; (g) Canny edge detection with double thresholding using HLS simulator; (h) Hardware based real time Canny edge detection result using OAA and OHET algorithms on Zynq 7000 AP SoC

The calculated gradient direction is used in the third stage of the Canny algorithm; NMS; which ensures that the final image showcases clear edges. The algorithm performs a complete scan of the image and identifies the pixels with the maximum value in the direction of the edge. The result is the image shown in Fig. 3f which includes edges with inconsistent intensities. This is because this stage does not discriminate between strong, genuine, and weak edges. Therefore, edges caused by noise or color variation are also preserved.

The result of the NMS stage is then processed through double thresholding, which removes "non-edges" from the image, i.e., pixels that are not relevant. The algorithm achieves this by using two threshold values. The threshold values distinguish between strong and weak pixels. A high threshold is employed to find strong pixels, i.e. pixels with an intensity gradient greater than a high threshold are designated as strong pixels. Conversely, pixels with an intensity gradient lower than the low threshold are deemed irrelevant. Lastly, pixels with an intensity gradient that fall between the high and low threshold values are identified as weak. The result of double thresholding, shown in Fig. 3g, displays several discontinuous edges. Weak pixels are then processed in the final step, hysteresis edge tracking.

C. Optimized Hysteresis Edge Tracking (OHET) Algorithm

The final step aims to convert the weak pixels found during double threshold into strong ones, but only if at least one of the neighboring pixels is a strong edge. This process helps in obtaining more continuous and connected edges. In this paper, we propose a depth-first search (DFS) algorithm shown in Fig. 4 to perform the final step. Starting from the strong pixel, the routine recursively probes its neighbors. If a neighbor is classified as a weak pixel, it is promoted to a strong edge. The pixel from which the edge tracing is started is first pushed into the stack. The routine then pops a pixel from the stack and examines its neighbors. For every neighboring pixel, the algorithm checks if the pixel is within bounds and is a weak edge. If both conditions are satisfied, then the pixel is promoted to a strong edge. The pixel is then pushed into the stack and then its neighbors are examined until there are no pixels left to examine.

Hysteresis Edge Tracking Algorithm

```
1: Input: image, startX, startY, width, height
2: Output: None
3:
4: procedure PromoteWeakEdges(image, startX, startY, width, height)
5:   Initialize an empty stack for pixel tracing
6:   Push the starting pixel (startX, startY) onto the stack
7:   while the stack is not empty do
8:     Pop the top pixel from the stack
9:     for each neighboring pixel do
10:      if the neighboring pixel is within bounds and is a weak edge then
11:        Promote the weak edge to a strong edge
12:        Push the neighboring pixel onto the stack
13:      end if
14:    end for
15:  end while
16: end procedure
17:
18: Algorithm HysteresisEdgeTracing(source, destination):
19: Constants: height ← IMAGE HEIGHT, width ← IMAGE WIDTH
20: for each pixel (x, y) in the image do end if
21:   if the pixel is a strong edge then
22:     PromoteWeakEdges(source, x, y, width, height)
```

23: ***end if***
24: ***end for***

Fig. 4. Proposed optimized hystersis edge tracking (OHET) algorithm

D. HLS Directives

Our implementation uses a line buffer and a sampling window that slides over the buffer to process the image. We utilize HLS directives, applying ARRAY_PARTITION to the SAMPLING_WINDOW array to enhance data access and reduce block RAM bottlenecks. For the LINE_BUFFER, we employ ARRAY_RESHAPE to optimize its layout, improving block RAM access without increasing usage. Additionally, we use the PIPELINE pragma to enable concurrent execution of several loops in the design.

IV. System Architecture And Implementation Results

The algorithm from HLS is converted and synthesized into a Xilinx IP and integrated into the system using IP integrator tool of Vivado. The entire system design can be seen in Fig. 5. To provide high-bandwidth direct memory access between the main memory (DDR) and the Canny edge detection IP, the system uses the AXI direct memory access (DMA) IP. The Zynq AP SoC comprises of processing system (PS) with ARM Cortex-A9 processor, and programmable logic (PL). The PL communicates with the PS through accelerated coherency port (ACP). The AXI to video and a video direct memory access (VDMA) IPs are used for transmitting video data and handling the transfer of processed video data.

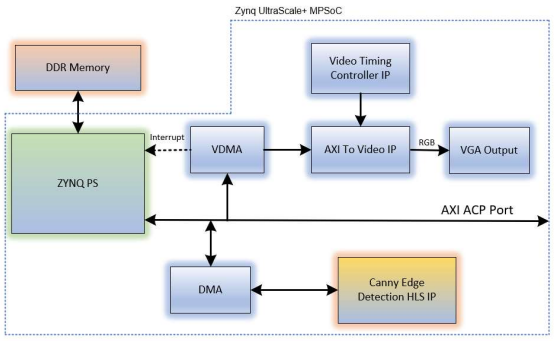

Fig. 5. Canny edge detection system block diagram

Table II shows the resource utilization for the complete Canny edge detection system shown in Fig. 5.

TABLE II. Canny Edge Detection Implementation Resource Utilization

Block	Sub-block	BRAM	DSP Blocks	FFs	LUTs
Canny Edge Detection	Gaussian Filter	2	0	1104	938
	Sobel Filter	1	3	1486	2157
	NMS	1	0	150	365
	Double Thresholding	0	0	50	117
	Hysteresis Edge Tracking	0	1	770	1577
	Total IP (Main)	94	5	3956	6267
VMDA	N/A	3	0	1941	2684
DMA	N/A	5	0	3099	2501
AXI to Video	N/A	1	0	161	254
Video Timing Controller	N/A	0	0	110	154
Glue Logic	N/A	0	0	2093	4076

Table III shows that our hardware acceleration approach, running at 100 MHz, achieves a speedup of 2.6 to 630 times in execution time for various image sizes compared to a software-based implementation on a single Cortex-A9 CPU running at 667 MHz.

TABLE III. Comparison Of Execution Time Between Propsoed Hardware Acceperated IP And Software Impelementation

IP Type	Size					
	64x64	128x128	256x256	312x312	512x512	1024x1024
Hardware accelerated Canny edge detection IP (ms)	9.34	9.42	10.28	10.46	11.49	14.04
Software based Canny edge detection IP (ms)	24.74	102.15	415.06	682.44	2020.78	8835.39

V. Conclusion

This paper presents a hardware acceleration of the Canny edge detection algorithm on a Xilinx Zynq 7000 AP SoC FPGA running at 100 MHz, achieving a speedup of 2.6-630x compared to a software implementation on a Cortex-A9 CPU at 667 MHz. This improvement was made using OAA and OHET algorithms for gradient direction and continuous edge formation. However, the OHET algorithm's recursive nature leads to high BRAM consumption, posing a design challenge. While our implementation is efficient in FF and LUT resources, optimizing the algorithm for streaming remains a future goal, as our design prioritizes FPGA area over memory.

References

[1] Canny, J., 1986. A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, (6), pp.679-698.

[2] https://www.xilinx.com/products/design-tools/vivado/high-level-design.html

[3] https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html

[4] Emrani, Z., Bateni, S. and Rabbani, H., 2017. A new parallel approach for accelerating the gpu-based execution of edge detection algorithms. Journal of Medical Signals & Sensors, 7(1), pp.33-42.

[5] Gentsos, C., Sotiropoulou, C.L., Nikolaidis, S. and Vassiliadis, N., 2010, December. Real-time canny edge detection parallel implementation for FPGAs. In 2010 17th IEEE International Conference on Electronics, Circuits and Systems (pp. 499-502). IEEE.

[6] Xu, Q., Varadarajan, S., Chakrabarti, C. and Karam, L.J., 2014. A distributed canny edge detector: algorithm and FPGA implementation. IEEE Transactions on Image Processing, 23(7), pp.2944-2960.

[7] Sangeetha, D. and Deepa, P., 2019. FPGA implementation of cost-effective robust Canny edge detection algorithm. Journal of Real-Time Image Processing, 16, pp.957-970.

[8] https://digilent.com/shop/zybo-z7-zynq-7000-arm-fpga-soc-development-board

[9] Sobel, I. and Feldman, G., 1968. A 3x3 isotropic gradient operator for image processing. a talk at the Stanford Artificial Project in, 1968, pp.271-272.

[10] Yamawaki, Akira, and Seiichi Serikawa, " A Describing Method of an Image Processing Software in C for a High-Level Synthesis Considering a Function Chaining," IEICE Transactions on Information and Systems, 2018.

[11] Woods, R.E., 2008. Rafael C. Gonzalez.

[12] Volder, J.E., 1959. The CORDIC trigonometric computing technique. IRE Transactions on electronic computers, (3), pp.330-334.