FPGA Implementation for Image Improved Edge Detection Algorithm Using Xilinx ISE Simulator

G. Sujatha[1], Avula Sushitha[2], VemuYayaathi Datta[3], Kamarthi Shashidhar[4], Marella Pavan Kumar[5] and Bairagi Kiranmayee[6]
{sujatha.g@svcolleges.edu.in[1], sushitha20@gmail.com[2], yayaathidatta2004@gmail.com[3], shashidharkamarthi@gmail.com[4], marellapavankumar00@gmail.com[5], lakshmibupathi273@gmail.com[6]}

Professor, Department of Electronics and Communication Engineering, Sri Venkateswara College of Engineering, Tirupati, Andhra Pradesh, India[1]
UG Scholar, Department of Electronics and Communication Engineering, Sri Venkateswara College of Engineering, Tirupati, Andhra Pradesh, India[2, 3, 4, 5, 6]

Abstract. The Canny edge detection algorithm is one of the most widely used methods for edge detection due to its superior performance. However, it is a complex and time-consuming process that also has a high hardware cost. In addition, most existing implementations use a fixed pair of high and low threshold values for all input images. Such fixed thresholds cannot automatically adapt to changes in the external detection environment, resulting in decreased performance. To address these issues, this paper proposes an improved Canny algorithm. It employs the Sobel operator and approximation methods to calculate the gradient magnitude and direction, thereby replacing complex operations and reducing hardware cost. Otsu's algorithm is introduced to adaptively determine the image threshold. Since Otsu's algorithm involves division operations that are slow and inefficient, this paper incorporates optimizations to improve efficiency. An optimized pipeline architecture is implemented for the Canny edge detection algorithm on an FPGA (Field Programmable Gate Array). The proposed method significantly enhances performance compared to the conventional approach, specifically on the FPGA device 7vx485tffg1157-3. The existing method, which used the conventional Canny algorithm with a pipeline architecture, achieved a delay of 6.061 ns (Maximum Frequency: 165.003 MHz) and consumed 1,908 Slice LUTs. In contrast, the proposed method reduces the delay to 3.897 ns (Maximum Frequency: 256.614 MHz) while utilizing only 889 Slice LUTs on the same FPGA device. These improvements are attributed to the optimized pipeline architecture, which streamlines data flow and computational stages. As a result, the proposed design demonstrates notable gains in both speed and resource utilization without compromising the accuracy of edge detection.

Keywords: Image processing, Canny edge detection algorithm, FPGA, Otsu's algorithm, logarithm approximation etc.

1 Introduction

Image processing is an essential topic used in medical diagnosis, computer vision, satellite picture and video surveillance. Among the various image-processing problems, edge detection is considered as one of the most significant processes because to determine object boundaries would help separate objects from backgrounds and detect intensity transitions. The Canny Edge Detection Algorithm One of the most widely used and successful algorithms is developed by John Canny in 1986. It became popular due to its good tradeoff between noise cancelation,

ICITSM-Part II 2025, April 28-29, Tiruchengode, India
Copyright © 2025 EAI
DOI 10.4108/eai.28-4-2025.2358041

localization and edge detection. Nevertheless, the Canny approach is relatively computational costly and not well suited for real-time implementation on traditional processors.

FPGAs have been put into service to ameliorate the limitation. In particular, FPGAs have the better power-performance when dealing with real-time image processing due to their ability of parallel computation and extreme low latency. However, the complexity of Canny algorithm such as square root and arctangent operations turns out to be challenging for its implementation on FPGA with limited resources. Moreover, it has fixed high and low thresholds of hysteresis that may perform poorly on images taken in different illuminations and contrasts.

To tackle these problems, we introduce an improved Canny Edge Detection Algorithm aimed at the FPGA-based realization using XILINX ISE as its software simulator. The advancements which include gradient magnitude and direction computation with both the Sobel operator and approximations, reduce computational over-heads. The Sobel filters are used to approximate computationally intensive mathematical operations, which lightens the computational load while preserving detection quality. Moreover, adaptive thresholding is achieved by using an efficient implementation of Otsu's approach with dynamic thresholds depending on the content of the processed image. This one minimizes expensive divisions as well, with benefit to speed and resources.

The proposed system uses the pipelined architecture in order to improve performance by allowing the parallel processing of blocks at different stages. This method increases throughput and decreases delay, allowing real-time processing of high-resolution images. The architecture was implemented and tested on Xilinx Virtex-7 FPGA (device: 7vx485tffg1157-3). The previous Canny approach exhibits better delay (3.1865 ns) at expense of a large number of Slice LUTs consumption (1324) compared with the classical one, also it needs an enormous area size (29162 slices), since we proposed method is more efficient in the sense that outperformed both traditional and previous one obtaining best results in terms of power with 3.897 ns of latency and consuming only 889 Slice LUTs for 2 steps implementing. These enhancements just prove the worth of hardware issues and algorithmic decision.

An FPGA-based parallel algorithm architecture is proposed as a computation acceleration and reconfigurable platform, which can achieve speedup more than twice the conventional solutions, and possesses reconfiguration facility, low-power property, and inter-image consistence. Applications that can be benefited from this system range from autonomous navigation, online surveillance, medicine image and embedded vision systems. The pipeline learning architecture and adaptive thersholding improve the robustness for different lighting, whereas approximations ease the hardware complexity with marginal accuracy penalizations. Since it is IP based, it can be easily implemented into more comprehensive image processing systems and expanded for future architectures.

The contributions of this paper can be summarized as follows:

- An approximation procedure which enables estimating both gradients magnitude and orientation.
- An adaptive threshold calculation of Otsu algorithm.

- A 32-bit logarithmic arithmetic unit for reducing the complexity of the Otsu thresholding technique.

The paper is arranged as follows:

In Section 2, a brief review of related works is provided, Section 3 discusses the existing methods, Section 4 states the proposed method, Section 5 presents and compares the simulation results for existing and proposed methods, and Section 6 concludes the paper.

2 Literature Review

2.1 Edge Detection Techniques

Kashyap et al. [15] also developed an edge detection method based on Sobel operator with median filter to enhance the noise robustness of images. Their result further recommands the merit of preprocessing using for performance boost on noisy environment [1]. Similarly, Mohammad et al. (2021), the Sobel edge detection method was tested with MATLAB and demonstrated how image features affect the sharpness of edges and efficiency of detection [2].

Elham et al. (2013) also applied the design study to Sobel edge detection, dealing with algorithm behavior on detecting image matching boundaries [4]. Meanwhile, Gonzalez et al. (2015) presented an extensive background for digital image processing, including classical edge detection models that would later act as the basis of FPGA implementations [8].

2.2 FPGA-Based Implementations of Edge Detection

Khidhir and Abdullah (2022) proposed an FPGA-based Sobel filter architecture with enhanced processing speed in comparison to the software approach used, with a conclusion that based on real-time image applications, FPGA is appropriate platform [3]. Anusha et al. (2012) then applied Sobel edge detection on FPGA, and demonstrated that hardware parallelism can support high-speed image processing [6].

In Abbasi and Abbasi (2007), an optimal FPGA architecture for the Sobel operator was proposed to keep resources highly utilized while producing correct results [7]. Xiangxi et al. (2018) expanded Sobel-based FPGA implementation to eight directions so that the accuracy and robustness of edge detection is improved in complicated images [9].

Brown et al. (2012) In his work, he introduces an in-depth study of the FPGAs and their architecture and their reconfigurability as they can be adapted to digital signal processing applications such as image edge detection [5].

2.3 Image Processing Fundamentals

In the textbook they authored, Gonzalez and Woods (2020) discussed fundamental concepts of digital image processing such as filtering, enhancing and edge detection, which are used in the theory design of FPGA [11].

Furthermore, the University of Tartu's Introduction to Image Processing is a pedagogical tool for basic methods (like filters and edge enhancement), as they provide theory and practical examples [10].

2.4 FPGA Tools and Hardware Platforms

The Xilinx resources [12] provide practical context for implementing image processing algorithms. The official Xilinx documentation explains FPGA devices and their applications, while the Digilent Zybo Z-10 [13] board information gives insight into hardware platforms often used for prototyping such designs.

3 Existing Method

As we know, the Canny edge detection algorithm is a standard method for edge detection, introduced by Canny, designed to compute the optimal edge detection by locating step edges in the presence of white Gaussian noise. We describe in Fig 1 for an input $M \times M$ size image for n-bit pixel depth, the algorithm makes several crucial steps. Firstly, the horizontal (Gx) and vertical (Gy) gradients, for each pixel, are computed by the convolution be applied gradient masks. Then, the gradient magnitude (G) and direction (θG) which describe the strength and orientation of the edge are obtained from these gradient components. The algorithm then performs Non-Maximum Suppression (NMS) to the thin edges by suppressing non-maximal points on the gradient direction, which are candidate edges. Then thresholding is applied by thresholding gradient magnitudes to high and low threshold values calculated from the gradient histogram of the image to make significant edge pixels. In the end, hysteresis thresholding connects edge segments by validating pixels that are between those thresholds, suppressing additional responses made by noises or lightening transitions. Although this approach works well in edge detection and noise elimination, it is computationally demanding especially for real time processing.

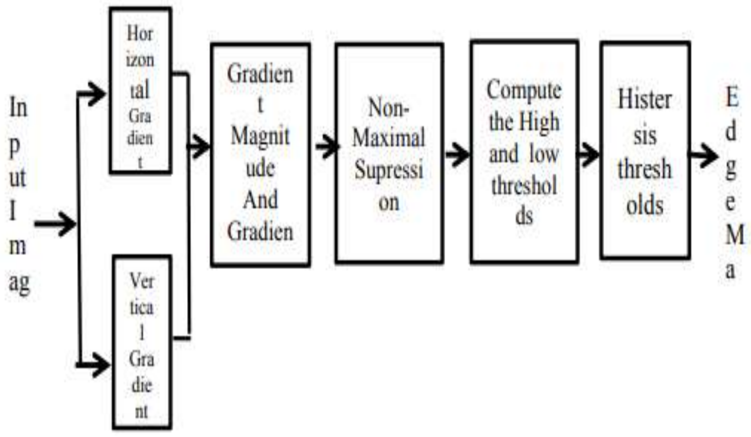


Fig. 1. Block diagram of the canny edge detection algorithm.

4 Proposed Method

4.1 Improved Canny Edge Detection Algorithm

Thresholding to specify edges in hardware: In the hardware- based Canny’s edge detection, the thresholding procedure holds an important role and basically, thresholding methods are categorized into two classes: fixed and adaptive. Hard thresholding drastically reduces the system design and hardware; however, it is less adaptable. It usually has to be recalibrated after each change of occular movements or specific aspect of external environmental features as object properties or lighting condition changes, and is not real timely appropriate. Such a limitation can be resolved if adaptive threshold is used since the algorithm will be able to automatically adapt to the changes in environmental conditions by itself.

In this work three modifications are proposed to make the Canny edge detector more hardware friendly. First, we incorporate Otsu's method for adaptive threshold selection so that the algorithm can automatically find the best threshold value for the image being processed. Then, a simplified computation for gradient magnitude/direction is employed to reduce the use of complex mathematical operations (e.g., square root, trigonometric functions) for lower hardware complexity. Third, we utilize logarithmic approximation to derive a simpler formulation of Otsu’s thresholding step, which often requires expensive divisions. These improvements collectively help to refactor the algorithm for real-time processing on FPGA platforms. Fig 2 shows the block diagram of the enhanced Canny edge detection flow. Each of these changes is described in further detail in the subsequent sections.

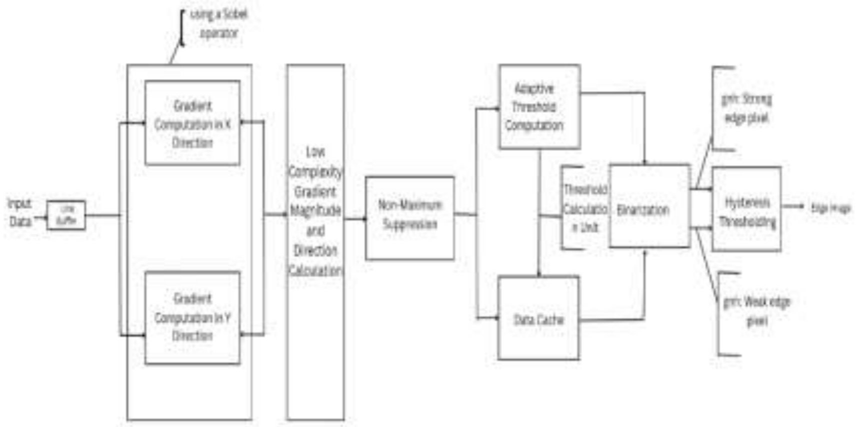

using a Sobel operator
Input Data
Line Buffer
Gradient Computation in X Direction
Gradient Computation in Y Direction
Low Complexity Gradient Magnitude and Direction Calculation
Non-Maximum Suppression
Adaptive Threshold Computation
Data Cache
Threshold Calculation Unit
Binarization
gm: Strong edge pixel
Hysteresis Thresholding
gm: Weak edge pixel
Edge image

Fig. 2. Architecture of improved canny edge detection.

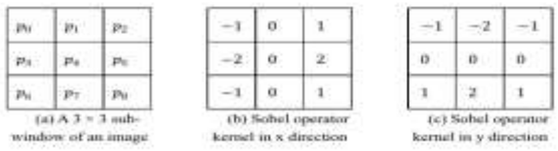
4.2 Gradient Computation Based on The Sobel Operator

In this project, the Sobel operator is chosen as the gradient mask for its combined ability to perform differentiation and smoothing, which helps reduce noise while emphasizing edge regions. This operator assigns higher weights to pixels near edges, improving the detection of important intensity changes. The gradient is computed using a 3 × 3-pixel window, as shown in Fig 3, where two Sobel kernels—one for horizontal (x) direction and one for vertical (y) direction—are applied. The horizontal and vertical gradients, denoted as fx(x,y) and fy(x,y), are calculated by convolving these kernels with the corresponding pixel values, using the equations given in (1) and (2).

To efficiently access pixel values for the convolution, a Line Buffer architecture is implemented, which stores the current and previous rows of pixel data using two FIFO modules, as illustrated in Fig 4. This buffering mechanism allows the system to retrieve the necessary 3 × 3 neighborhood pixels without delay during image scanning.

$$f_x(x,y) = (p_2 - p_0) + 2(p_5 - p_3) + (p_8 - p_6) \quad (1)$$

$$f_y(x,y) = (p_6 - p_0) + 2(p_7 - p_1) + (p_8 - p_2) \quad (2)$$



p_0	p_1	p_2
p_3	p_4	p_5
p_6	p_7	p_8

(a) A 3 × 3 sub-window of an image

−1	0	1
−2	0	2
−1	0	1

(b) Sobel operator kernel in x direction

−1	−2	−1
0	0	0
1	2	1

(c) Sobel operator kernel in y direction

Fig. 3. Sub-window of an image and Sobel operator kernel.

Once pixels are retrieved, the gradient components fx(x,y) and fy(x,y), are computed through hardware modules designed to perform weighted sums using multipliers and adders, as depicted in Fig 5 for the horizontal gradient. The vertical gradient calculation uses a similar architecture with different kernel weights. Both gradient outputs are represented as signed numbers, where the most significant bit (MSB) indicates the sign of the gradient value, essential for later stages such as direction calculation and edge thinning.

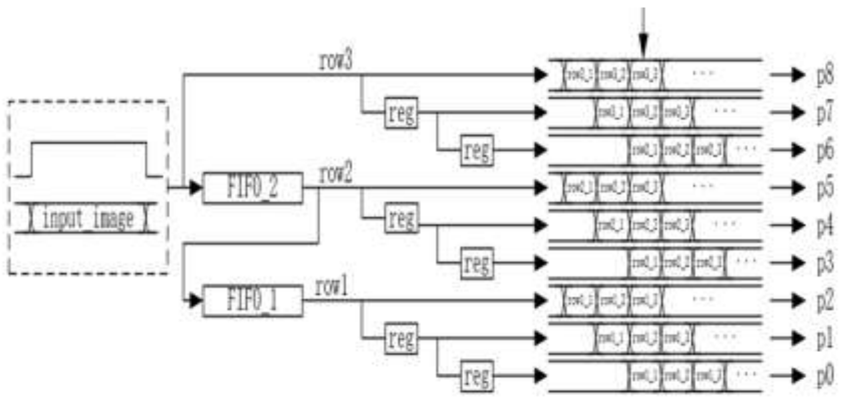


Fig. 4. Architecture of Line Buffer.

This combination of the Sobel operator and the line buffer design provides an efficient balance between computational complexity and hardware resource usage, enabling accurate and real-time gradient computation as a crucial step in the edge detection process.

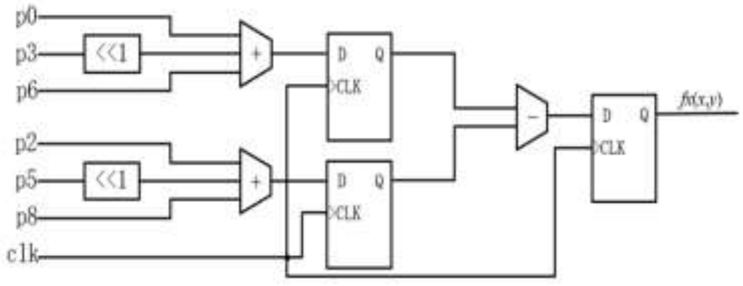


Fig. 5. Architecture of fx(x, y).

4.3 Low Complexity Gradient Magnitude and Direction Calculation

In image processing, accurately determining the gradient magnitude and direction at each pixel is critical for effective edge detection. The gradient magnitude represents the strength or intensity of the edge at a particular pixel location, while the gradient direction provides information about the orientation of the edge. Conventionally, the gradient magnitude and direction are computed using well-known mathematical formulas, which involve square root and arctangent operations. Specifically, the magnitude at pixel coordinates (x, y) is calculated as the square root of the sum of squares of the horizontal and vertical gradients.

$$mag(x,y) = \sqrt{f_x(x,y)^2 + f_y(x,y)^2} \tag{3}$$

$$\theta(x,y) = \arctan\left(\frac{f_y(x,y)}{f_x(x,y)}\right) \tag{4}$$

To mitigate these issues, this study adopts a simplified approach for gradient magnitude calculation that avoids expensive square root operations. Instead of computing the exact Euclidean magnitude, we approximate the gradient magnitude by summing the absolute values of the horizontal and vertical gradient components, as suggested in several prior works. This approximation allows the magnitude calculation expressed as,

$$mag(x,y) = |f_x(x,y)| + |f_y(x,y)| \tag{5}$$

While the simplification of magnitude calculation is straightforward, determining the gradient direction is more challenging. Traditional methods often use the Coordinate Rotation Digital Computer (CORDIC) algorithm to calculate the arctangent function, which is required to find the gradient orientation. Although the CORDIC algorithm is a hardware-friendly iterative method for computing trigonometric functions, it requires multiple clock cycles and iterations to reach the desired accuracy, leading to increased hardware complexity and latency. These factors make CORDIC less suitable for real-time image processing applications where speed and resource efficiency are critical.

To overcome the limitations of the CORDIC method, this project implements a novel, shift-based technique for gradient orientation calculation. Instead of calculating the precise arctangent value, the orientation space is divided into eight distinct zones, each representing a range of edge directions distributed evenly across 360 degrees. Fig 6 illustrates these eight orientation zones, with each zone corresponding to a specific angular sector. This method approximates the continuous range of gradient directions by categorizing them into discrete intervals, significantly simplifying the hardware implementation. Fig.7 shows the Architecture of the gradient magnitude calculation and quadrant flag computation.

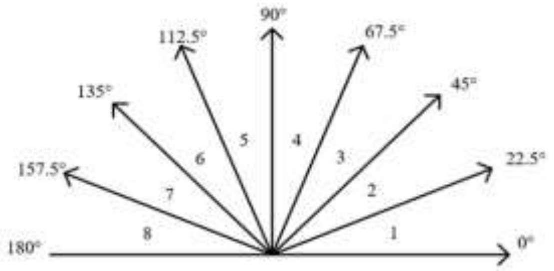


Fig. 6. Eight zones of gradient orientation**.**

To identify the correct zone for θ (x, y), we adjust θ (x, y) to tanθ(x, y), where each zone corresponds to a specific range between two angle values. For simplicity, Equation (4) can be rewritten as follows:

$$\tan\theta\,(x,y) = \frac{f_y(x,y)}{f_x(x,y)} \tag{6}$$

The value of tanθ(x, y) is given by the Equation (7).

$$\tan\theta_j\,(x,y) \le \tan\theta\,(x,y) < \tan\theta_{j+1}\,(x,y) \tag{7}$$

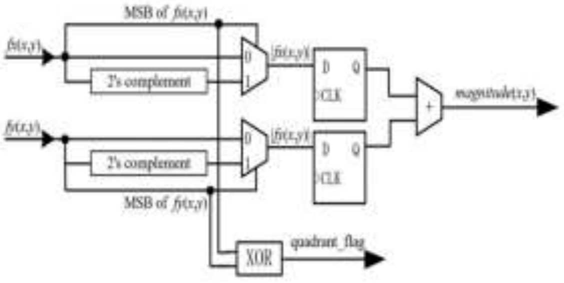


Fig.7. Architecture of the gradient magnitude calculation and quadrant flag computation**.**

Rewriting the Equation (7) gives us Equation (8)

$$\tan \theta_j\,(x,y) \le \frac{f_y(x,y)}{f_x(x,y)} < \tan \theta_{j+1}\,(x,y) \tag{8}$$

The condition derived from Equation (8) is given in Equation (9)

$$f_x(x,y) * \tan \theta_j\,(x,y) \le f_y(x,y) < f_x(x,y) * \tan \theta_{j+1}\,(x,y) \tag{9}$$

The value of zone is determined by satisfying the condition specified in Equation (9). Fig 8 illustrates the architecture used to calculate fx(x, y) * tan22.5° and to identify the corresponding direction of the zone.

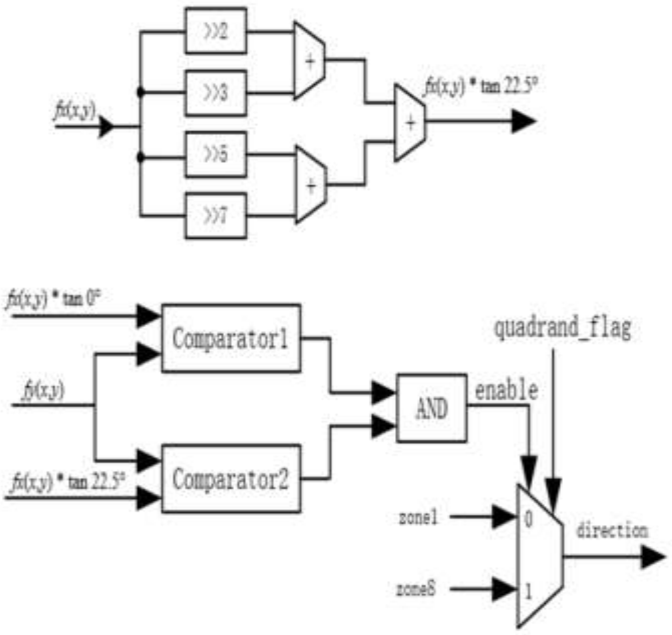


Fig. 8. Architecture to find the direction.

4.4 Non-Maximum Suppression

Non-maximum suppression (NMS) is a technique used to thin detected edges by preserving only the local maxima in the direction of the image gradient. The architecture for NMS is illustrated in Fig 9. In this process, a selector determines the two neighboring pixels along the gradient direction relative to the current pixel (which is the center pixel in a 3 × 3 neighborhood), denoted as pixels **a** and **b**. For gradient directions labeled 1 or 8, pixels **a** and **b** correspond to positions g3 and g5; for directions 2 or 3, they are g0 and g8; directions 4 or 5 correspond to g2 and g6; and directions 6 or 7 correspond to g1 and g7. A comparator then evaluates whether the current pixel value (g4) is greater than both neighboring pixels **a** and **b** to determine if it is a local maximum. If it is the local maximum, the pixel is retained; otherwise, it is suppressed by setting its value to zero. To optimize hardware usage, although the pixel intensity g4 can be represented with up to 10 bits, the value is limited to 8 bits, thereby conserving resources for later stages in the processing pipeline.

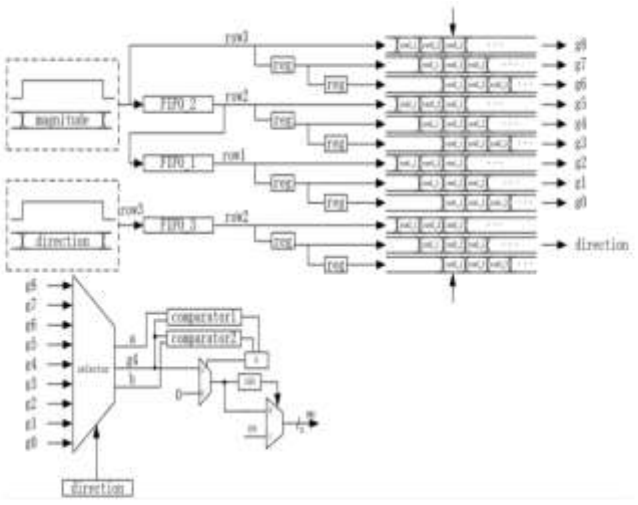


Fig. 9. Architecture of NMS.

4.5 Adaptive Threshold Computation

Adaptive thresholding is a crucial component in the enhanced Canny edge detection approach proposed. Following the Non-Maximum Suppression (NMS) stage, Otsu's method is employed to dynamically determine the optimal threshold values. Since thresholding requires analysis of the entire image's data, the NMS output is cached with an SDRAM controller as an interim step to this decoding. The histogram of the NMS data is calculated so that one can run Otsu's algorithm to determine an adaptive threshold value. Once this threshold is achieved, the system reads the NMS data from the SDRAM and compares it with the determined threshold, representing the data in binary. Then, hysteresis thresholding is enforced on the edge detection result. This order makes it possible to support flexible threshold according to an image property, enhancing the edge detection accuracy and robustness.

4.5.1 32-Bit Logarithmic Arithmetic Unit

The LNS is investigated as an alternative to reduce arithmetic operations and computational complexity [26–28]. The system is composed of a number of key modules: a 32-bit CLZ/Leading Zero unit, a BSH, a CGen and a xFPGen. The input number in the Qm. n format to a fixed Q6. 26 formats by logarithm transfer. The number of leading zeros is encoded in the CLZ block and exported as a five-bit result which is utilized to obtained both the characteristic value and the shift amount in the Barrel Shifter. The Barrel Shifter then modifies the mantissa range for Qm. n to the Q6. 26 formats. Then the fractional part is generated by the FPGen, where the characteristic part is generated by the CGen. The resulting two parts are then added together to create the overall output of the log conversion. Fig 10 shows the Architecture of logarithmic converter.

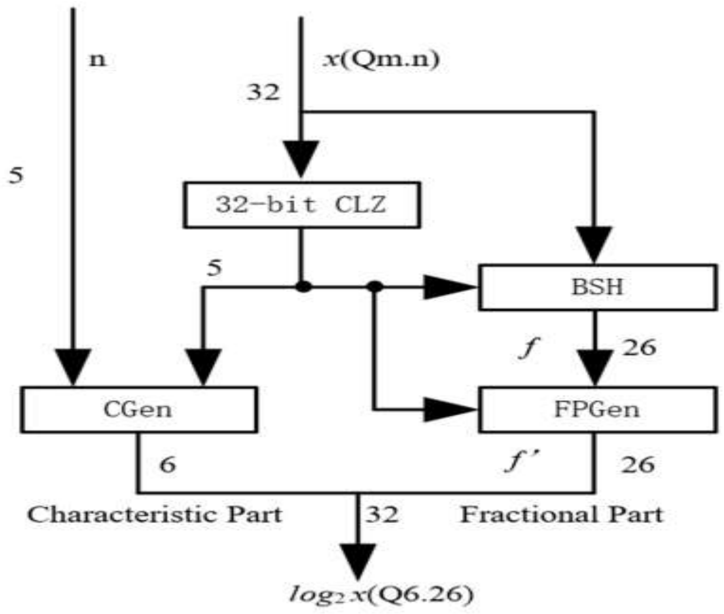


Fig. 10. Architecture of logarithmic converter**.**

5 Simulation Results

5.1 Existing System

Fig 11 and fig 12 shows the Schematic Diagram of Existing Canny Edge RTL design.

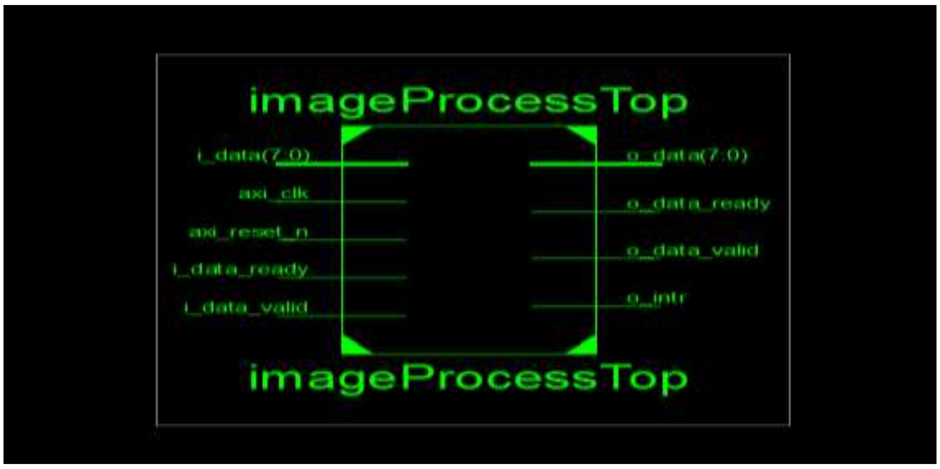


Fig. 11. Schematic Diagram of Existing Canny Edge RTL design.

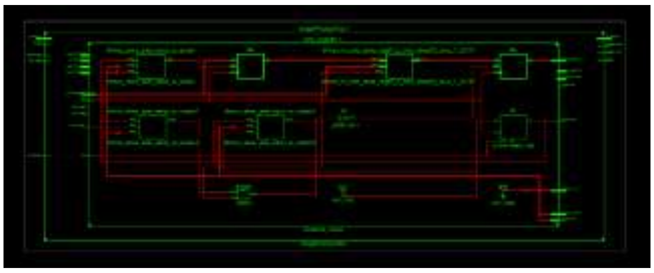
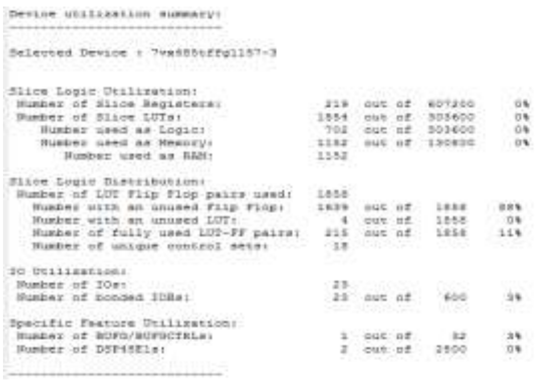


Fig. 12. Schematic Diagram of Existing Canny Edge RTL design.

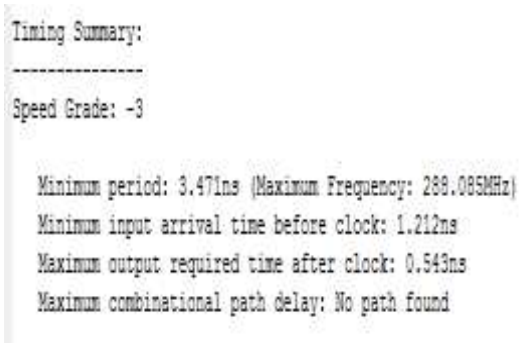
Fig 13 shows the Area Utilization of Existing Canny Edge RTL design.



```
Device utilization summary:
---------------------------

Selected Device : 7vx585tffg1157-3


Slice Logic Utilization:
 Number of Slice Registers:              219  out of  607200     0%
 Number of Slice LUTs:                  1854  out of  303600     0%
    Number used as Logic:                702  out of  303600     0%
    Number used as Memory:              1152  out of  130800     0%
       Number used as RAM:              1152

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    1858
   Number with an unused Flip Flop:     1639  out of    1858    88%
   Number with an unused LUT:              4  out of    1858     0%
   Number of fully used LUT-FF pairs:    215  out of    1858    11%
   Number of unique control sets:         18

IO Utilization:
 Number of IOs:                           23
 Number of bonded IOBs:                   23  out of     600     3%

Specific Feature Utilization:
 Number of BUFG/BUFGCTRLs:                 1  out of      32     3%
 Number of DSP48E1s:                       2  out of    2800     0%

---------------------------
```

Fig. 13. Area Utilization of Existing Canny Edge RTL design.

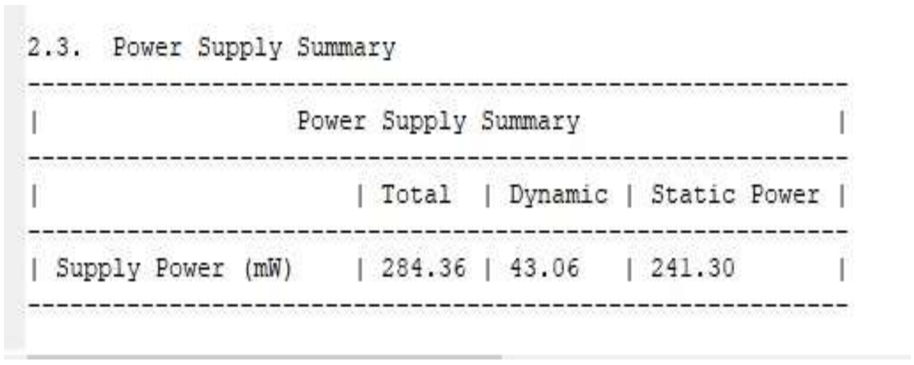
Fig 14 shows the Delay of Existing Canny Edge RTL design.



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 3.471ns (Maximum Frequency: 288.085MHz)
   Minimum input arrival time before clock: 1.212ns
   Maximum output required time after clock: 0.543ns
   Maximum combinational path delay: No path found
```

Fig. 14. Delay of Existing Canny Edge RTL design.

Fig 15 shows the Power consumption of Existing Canny Edge RTL design.



```
2.3.  Power Supply Summary
----------------------------------------------------------
|                  Power Supply Summary                  |
----------------------------------------------------------
|                      | Total  | Dynamic | Static Power |
----------------------------------------------------------
| Supply Power (mW)    | 284.36 | 43.06   | 241.30       |
----------------------------------------------------------
```

Fig. 15. Power consumption of Existing Canny Edge RTL design.

Fig 16 shows the Output Wave form for Existing Canny Edge RTL design

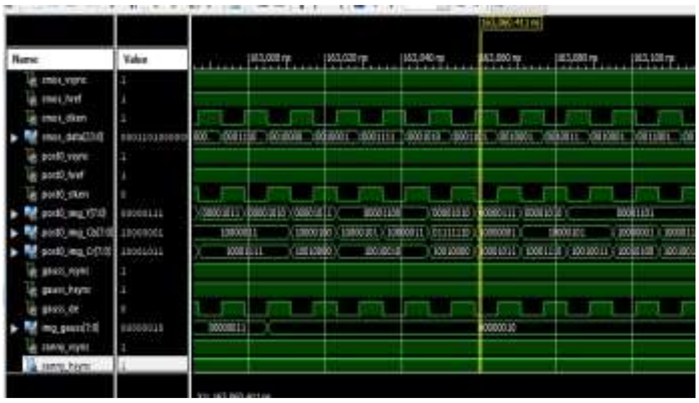


Fig. 16. Output Wave form for Existing Canny Edge RTL design.

5.2 Proposed Method

Fig 17 and fig 18 shows the Schematic Diagram of Proposed Canny Edge RTL design.

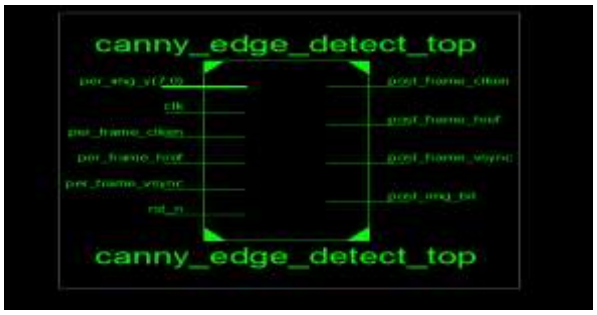


Fig. 17. Schematic Diagram of Proposed Canny Edge RTL design.

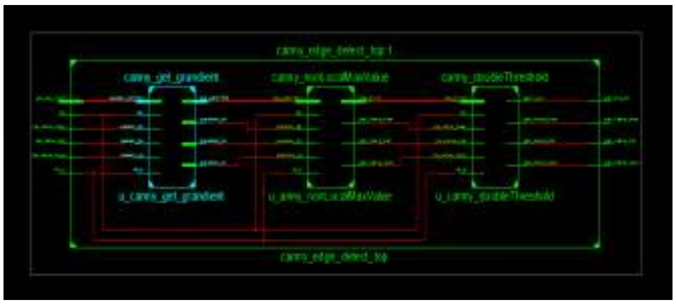
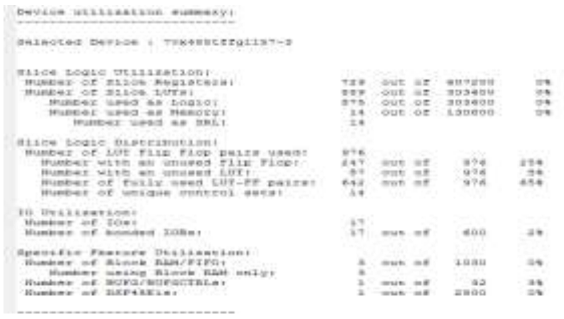


Fig. 18. Schematic Diagram of Proposed Canny Edge RTL design.

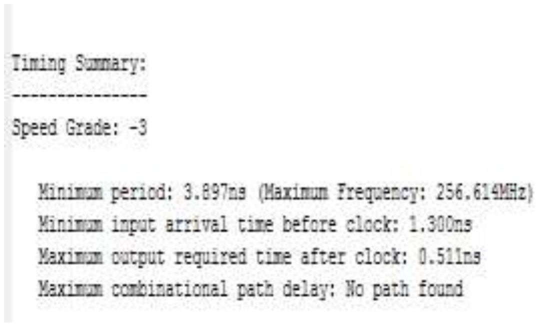
Fig 19 shows the Area Utilization of Proposed Canny Edge RTL design.



```
Device utilization summary:
---------------------------

Selected Device : 7vx485tffg1157-3

Slice Logic Utilization:
 Number of Slice Registers:              729  out of  607200     0%
 Number of Slice LUTs:                   889  out of  303600     0%
    Number used as Logic:                875  out of  303600     0%
    Number used as Memory:                14  out of  130800     0%
       Number used as SRL:                14

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:     976
   Number with an unused Flip Flop:      247  out of     976    25%
   Number with an unused LUT:             87  out of     976     8%
   Number of fully used LUT-FF pairs:    642  out of     976    65%
   Number of unique control sets:         14

IO Utilization:
 Number of IOs:                           17
 Number of bonded IOBs:                   17  out of     600     2%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                 8  out of    1030     0%
    Number using Block RAM only:           8
 Number of BUFG/BUFGCTRLs:                 1  out of      32     3%
 Number of DSP48E1s:                       1  out of    2800     0%

---------------------------
```

Fig. 19. Area Utilization of Proposed Canny Edge RTL design.

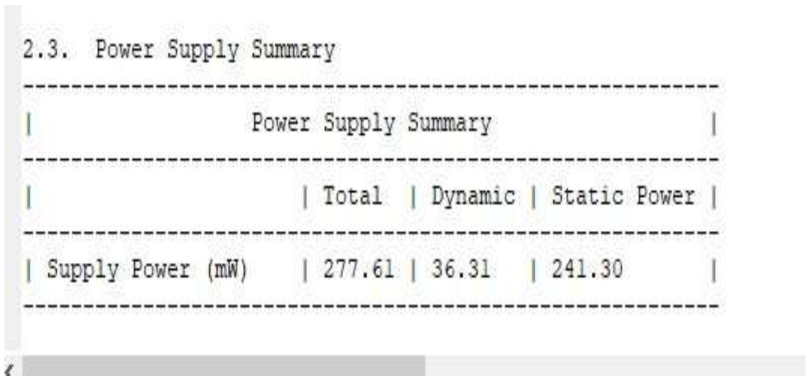
Fig 20 shows the Delay of Proposed Canny Edge RTL design.



```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 3.897ns (Maximum Frequency: 256.614MHz)
   Minimum input arrival time before clock: 1.300ns
   Maximum output required time after clock: 0.511ns
   Maximum combinational path delay: No path found
```

Fig. 20. Delay of Proposed Canny Edge RTL design.

Fig 21 shows the Power consumption of Proposed Canny Edge RTL design.



```
2.3.  Power Supply Summary
-----------------------------------------------------------
|                  Power Supply Summary                   |
-----------------------------------------------------------
|                     | Total  | Dynamic | Static Power |
-----------------------------------------------------------
| Supply Power (mW)   | 277.61 | 36.31   | 241.30       |
-----------------------------------------------------------
```

Fig. 21. Power consumption of Proposed Canny Edge RTL design.

Fig 22 shows the Output Wave form for Proposed Canny Edge RTL design.

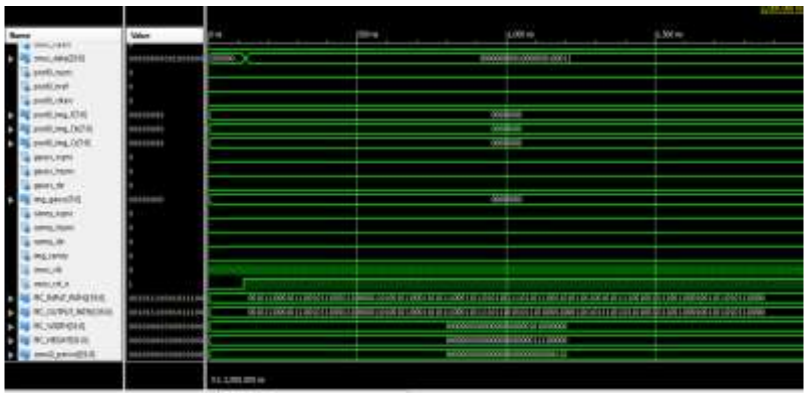


Fig. 22. Output Wave form for Proposed Canny Edge RTL design.

5.3 Results of Discussions

Table 1. Results Comparison.

S. No	Parameter	Existing Method	Proposed Method
1	Area in LUT's	1854	889
2.	Delay (in ns)	3.741	3.897
3.	Max. Frequency (in MHz)	288.085	256.614
4	Power Delay Product (PDP) (in W/sec)	1063.790	1081.846
5.	Power (in Milli Watts)	284.36	277.61

The table 1 above presents a comparative evaluation of device performance for various designs following synthesis on the Xilinx Vertex7 – 7vx485tfgg1157 FPGA board.

The proposed architectures were developed using Verilog HDL and simulated with grayscale images of varying dimensions, employing a fixed threshold value for testing purposes.

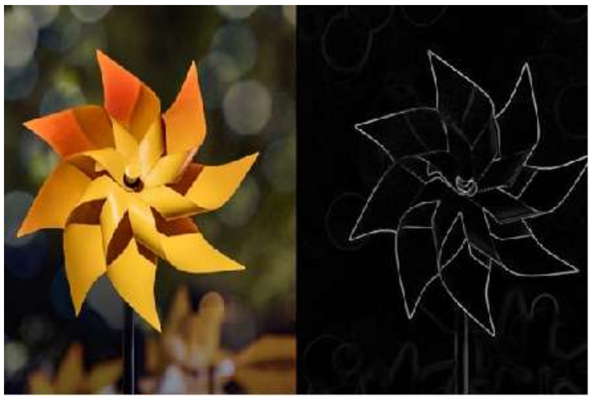
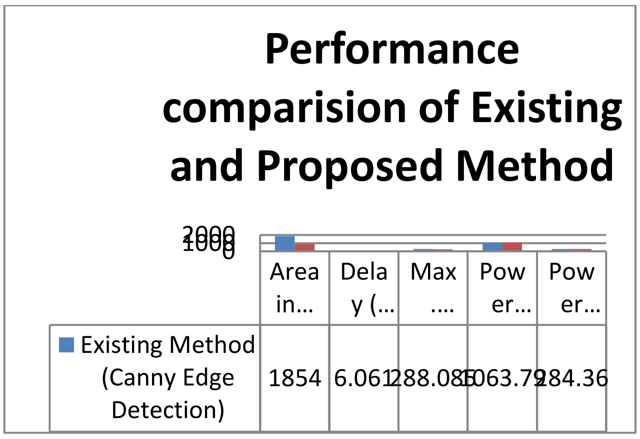


Fig. 23. Corresponding edge images by hardware and software package implementation.

Fig 23 illustrates an 8-bit input image alongside the resulting edge-detected outputs produced by software implementations. A threshold intensity of 27 was consistently applied in both methods. The design functions at a clock frequency of 100 MHz, and Table 1 provides a comparison of execution times across different architectures. According to synthesis results, the design utilized 889 logic cells and covered an area of approximately 1951.288 μm². Additionally, it consumed about 277.61 mW of power, with a propagation delay measured at 9.599 ns. Fig. 24 shows the Performance of Existing and Proposed methods.

5.4 Performance Comparison



	Area in…	Dela y (…	Max ….	Pow er…	Pow er…
🟦 Existing Method (Canny Edge Detection)	1854	6.061	288.085	1063.79	284.36

Fig. 24. Performance of Existing and Proposed methods.

6 Conclusion

This study presents an in-depth implementation of the Canny Edge Detection Algorithm on an FPGA using a pipeline architecture. Comparative analysis shows that the proposed design substantially lowers the delay, achieving a minimum clock period of 3.897ns (corresponding to a maximum frequency of 256.614MHz) while using only 889 Slice LUTs on the FPGA model 7vx485tffg1157-3. This represents a significant enhancement compared to the earlier design, which experienced a delay of 6.061ns (maximum frequency of 165.003MHz) and consumed 1908 Slice LUTs on the same device. These improvements in timing performance and resource efficiency highlight the effectiveness of the optimized pipeline architecture, making it well-suited for real-time edge detection tasks on FPGA platforms. The findings clearly demonstrate that the refined pipeline structure is key to accelerating edge detection without compromising hardware utilization.

7 Future Scope

Future research may aim at further optimizing the Canny Edge Detection Algorithm to better align with FPGA hardware capabilities. This could involve fine-tuning specific stages of edge detection or exploring alternative algorithms that have the potential to deliver even greater performance enhancements.

References

[1] Kashyap, Y., Vyas, A., Raghuwanshi, R., & Sharma, R. (2015). Edge detection using Sobel method with median filter. *International Journal of Modern Trends in Engineering and Research, 2*(5), 372–378. Retrieved from https://scispace.com/pdf/edge-detection-using-sobel-method-with-median-filter-468qtersj7.pdf

[2] Mohammad, E. J., JawadKadhim, M., Hamad, W. I., & Helyel, S. Y. (2014). Study Sobel edge detection effect on the image edges using MATLAB. *International Journal of Innovative Research in Science, Engineering and Technology, 3*(3). Retrieved from https://www.ijirset.com/upload/2014/march/68_Study.pdf

[3] Jadhav, G., Yadav Jada, S., Triveni, R., & Khan, S. I. (2021, July). FPGA based edge detection using Sobel filter. International Journal for Research in Applied Science & Engineering Technology (IJRASET), 9(VII), 145–147. https://doi.org/10.22214/ijraset.2021.36276

[4] Elham, J. M., Ahmed, J. M., Zainab, J. M., et al. (2013). Design study Sobel edge detection**.** *International Journal of Application or Innovation in Engineering & Management (IJAIEM), 2*(12), 248–253. Retrieved from https://studylib.net/doc/13268361/international-journal-of-application-or-innovation-in-

[5] Brown, S. D., Francis, R. J., Rose, J., & Vranesic, Z. G. (2012). *Field-programmable gate arrays*. Springer. https://doi.org/10.1007/978-1-4615-3572-0

[6] Anusha, G., JayaChandra Prasad, T., & SatyaNarayana, D. (2012). Implementation of Sobel edge detection on FPGA. *International Journal of Computer Trends and Technology, 3*(3). Retrieved from https://www.ijcttjournal.org/Volume3/issue-3/IJCTT-V3I3P127.pdf

[7] Abbasi, T. A., & Abbasi, M. U. (2007). A proposed FPGA-based architecture for Sobel edge detection operator. *Journal of Active and Passive Electronic Devices, 2*, 271–277. Retrieved fromhttps://www.researchgate.net/publication/254846702_A_Proposed_FPGA_Based_Architecture_for_Sobel_Edge_Detection_Operator

[8] Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2020). *Digital image processing using MATLAB*. MathWorks. Retrieved from https://www.mathworks.com/academia/books/digital-image-processing-using-matlab-gonzalez.html

[9] Z. Xiangxi, Z. Yonghui, Z. Shuaiyan and Z. Jian, "FPGA implementation of edge detection for Sobel operator in eight directions," *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Chengdu, China, 2018, pp. 520-523, doi: 10.1109/APCCAS.2018.8605703.

[10] University of Tartu. *Introduction to image processing* (Textbook 1). Retrieved from https://sisu.ut.ee/imageprocessing/1-2/

[11] Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing* (4 e.). Pearson. Retrieved from https://www.mathworks.com/academia/books/digital-image-processing-gonzalez.html

[12] "Xilinx Products" https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[13] DigilentZybo Z-10 Product Information" https://www.xilinx.com/products/boards-and-kits/1-pukimv.html.