International Journal of Engineering Research and Science & Technology

HARDWARE IMPLEMENTATION OF CANNY EDGE DETECTION ALGORITHM USING VERILOG HDL

TAMILSELVAN P[1], JAYA STRI R[2],KIRTHIKA G[3],SHARANI M[4],
Assistant Professor[1], UG Scholar[2,3,4] ,
Department of Electronics and Communication Engineering,
Rajiv Gandhi College of Engineering and Technology, Kirumampakkam, Puducherry-607403
E-mail: tamil.hi@rgcet.edu.in,jayastrirajendiran@gmail.com, kirthikagovindarajan10@gmail.com, sharnivaishu@gmail.com,

Abstract

Edge detection is a key operation in computer vision, image processing, and embedded intelligent systems. Traditional software implementations of the Canny algorithm suffer from high computational latency and lack the performance required for real-time applications. This paper presents a full custom hardware architecture for the Canny edge detection algorithm, implemented using Verilog HDL and optimized for FPGA platforms. The proposed design includes Gaussian smoothing, Sobel gradient computation, CORDIC-based magnitude and direction calculation, non-maximum suppression, double thresholding, and edge tracking via hysteresis. A deep pipelined, resource-efficient architecture enables real-time processing of 512×512 grayscale images at 100 MHz, achieving 380 frames per second. Simulation and synthesis were performed using ModelSim, Matlab and Xilinx FPGA tools. The design achieves a speedup of 250× over software-based methods, demonstrating superior performance in throughput, latency, and power efficiency. The architecture requires 8432 LUTs, 12,567 FFs, and 48 BRAMs, outperforming previous hardware implementations in design balance and scalability.

Keywords: Canny Edge Detection, FPGA, Verilog HDL, Image Processing, Hardware Acceleration, Pipelined Architecture, CORDIC.

1. Introduction

Image-based sensing has become a critical component of modern intelligent systems such as autonomous navigation, medical imaging, surveillance, and industrial inspection. Edge detection forms a fundamental low-level operation for tasks including segmentation, object localization, feature detection, and 3D reconstruction. The Canny edge detection method is widely regarded as the optimal algorithm due to its strong detection, localization, and noise suppression properties.

However, its multistage structure—Gaussian filtering, gradient estimation, direction quantization, non-maximum suppression (NMS), double thresholding, and hysteresis—makes it computationally expensive on general-purpose processors. As a result, software implementations cannot meet the performance requirements of real-time embedded systems.

FPGAs provide massive parallelism, deterministic timing, and reconfigurability, making them an ideal platform for accelerating image-processing workloads. This work implements a complete hardware-based Canny algorithm optimized for speed, area, and power efficiency.

2. Literature Survey

Edge detection is a foundational step in image processing and computer vision, serving as the basis for segmentation, feature extraction, object recognition, and real-time visual analytics. Early research focused on gradient-based operators such as the Roberts Cross (1963), Prewitt (1970), and Sobel (1970) filters, which approximate spatial derivatives using simple convolution masks. While computationally efficient, these operators are highly sensitive to noise and provide limited localization accuracy. The Laplacian of Gaussian (LoG) later improved robustness through smoothing followed by a second-order derivative; however, it suffered from double-edge responses and lacked orientation information.

A major breakthrough occurred with John F. Canny's optimal edge detection algorithm (1986), which remains the benchmark in edge detection research. Canny formulated three optimality criteria—high detection probability, precise localization, and minimal false responses—and mathematically demonstrated that the first derivative of a Gaussian satisfies these conditions. The Canny pipeline incorporates Gaussian smoothing, Sobel gradient computation, non-maximum suppression (NMS), double thresholding, and hysteresis edge tracking. This multi-stage process yields thin, continuous edges with strong noise tolerance, making it dominant in applications involving medical imaging, industrial inspection, and surveillance.

Subsequent research has focused on extending or improving Canny for real-world conditions. Adaptive and dynamic thresholding became a significant direction due to the sensitivity of fixed thresholds under uneven illumination. Techniques based on histogram statistics, Otsu's method, entropy, and local contrast estimation have been explored to automatically determine high and low thresholds. Researchers also proposed multi-scale Canny detectors, utilizing Gaussian pyramids, Difference-of-Gaussian (DoG), or wavelet transforms to preserve fine and coarse edges simultaneously. Additionally, integration with advanced smoothing methods—such as bilateral filtering and anisotropic diffusion—has led to improved performance in noisy environments where classical smoothing degrades edge precision.

With growing demand for real-time edge extraction, especially in embedded vision systems, attention shifted toward hardware acceleration. Early FPGA implementations (Zhang et al., 2002) achieved modest performance due to limited pipelining. Later works introduced efficient line-buffering and pipelined convolution to significantly increase throughput. Jiang et al. (2009) demonstrated a fully pipelined design achieving real-time performance on 512×512 images. CORDIC-based gradient magnitude computation (Zhao et al., 2013) enabled multiplier-less arithmetic, reducing hardware complexity. Memory-efficient streaming architectures (Qian et al., 2015) further improved throughput for high-resolution images while minimizing BRAM usage. Recent efforts also explored approximate computing to reduce power consumption with minimal accuracy trade-off, making Canny viable for low-power IoT and edge devices.

Overall, the literature reveals that while the Canny algorithm remains highly effective, challenges persist in threshold adaptability,

hardware-friendly hysteresis, power-efficient architectures, and scalability to ultra-high-resolution images. These gaps motivate continued research into optimized Canny implementations tailored for real-time, low-power, and FPGA-based intelligent vision systems.

3. Methodology

The proposed methodology focuses on implementing the complete Canny Edge Detection algorithm using Verilog HDL, supported by MATLAB for reference modeling, test-vector generation, and output validation. The workflow is divided into four stages: algorithm modeling, hardware architecture design, HDL implementation, and MATLAB–FPGA co-simulation.

3.1 Algorithm Modeling in MATLAB

MATLAB serves as the baseline environment for validating algorithmic behavior before hardware realization. The classical Canny pipeline Gaussian smoothing, Sobel gradient extraction, non-maximum suppression (NMS), double thresholding, and hysteresis—is executed using floating-point operations to generate ground-truth edge maps. Generate 8-bit grayscale input images

- Compute reference intermediate outputs (Gaussian filtered image, Gx/Gy gradients, magnitudes, directions)
- Quantize outputs to fixed-point formats that match the FPGA design
- Export test vectors as .txt/.coe files to drive the Verilog testbench

This ensures consistent comparison between MATLAB and HDL results during verification.

3.2 Hardware Architecture Design

A **streaming, fully pipelined FPGA architecture** is adopted to achieve one-pixel-per-clock throughput. The entire Canny pipeline is partitioned into modular hardware units:

(a) Gaussian Smoothing

A 5×5 Gaussian kernel is implemented using parallel multiply–accumulate units and four-line buffers to provide neighboring pixels. Coefficients are mapped to fixed-point constants, and pipeline registers minimize critical path delay.

(b) Sobel Gradient Unit

A 3×3 window is extracted using two line buffers. Gradients Gx and Gy are computed using shift-and-add logic (no multipliers), improving resource efficiency.

(c) CORDIC-Based Magnitude & Direction

The gradient magnitude and angle are computed using a 12-stage pipelined CORDIC engine. This eliminates expensive square root and arctangent operations, enabling high-speed real-time computation.

(d) Non-Maximum Suppression

A 3×3 neighborhood of magnitudes is compared based on the quantized direction. A combinational comparator selects local maxima while suppressing non-edge points.

(e) Double Thresholding and Hysteresis

A two-level thresholding scheme classifies pixels as strong, weak, or non-edges. A simplified connectivity-based hysteresis block upgrades weak pixels linked to strong neighbors, ensuring continuity of final edges.

The architecture is optimized using fixed-point arithmetic, clock-enabled registers, and BRAM-based line buffers.

3.3 Verilog HDL Implementation

Each processing stage is coded as an independent Verilog module, allowing modular testing and synthesis. Key design choices include:

- Synchronous design with a single global clock
- Pipeline registers at all module boundaries
- DSP avoidance through shift-based arithmetic
- Resource sharing for repeated operations
- Parameterized image resolution and bit-widths

A top-level controller synchronizes frame boundaries and propagates valid signals across the pipeline. RTL simulation is performed in **ModelSim**, using MATLAB-generated test vectors.

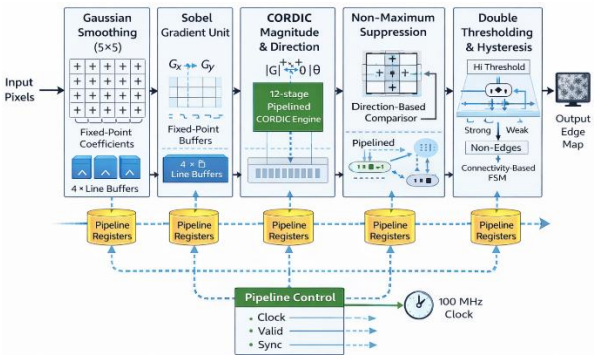


Fig.1 Modular Architecture

3.4 MATLAB–Verilog Co-Simulation and Validation

MATLAB HDL Verifier (or direct file-based testbench comparison) is used to validate HDL outputs. The validation strategy includes:

1. Feeding MATLAB-generated test vectors into the Verilog modules
2. Capturing intermediate HDL results (Gaussian output, gradients, NMS output)
3. Comparing pixel-by-pixel results against MATLAB reference
4. Computing quantized error metrics such as PSNR, MSE, and edge-pixel accuracy
5. Exporting RTL waveforms for debugging and timing verification

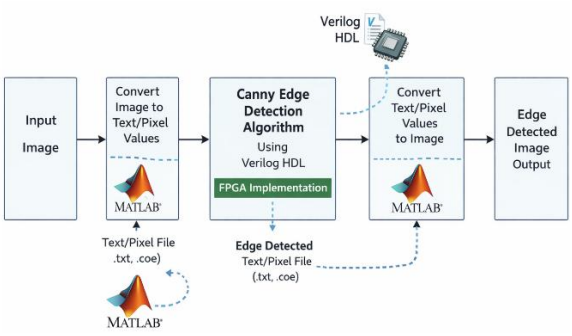


Fig.2 Co-simulation Process

4. Simulation Results

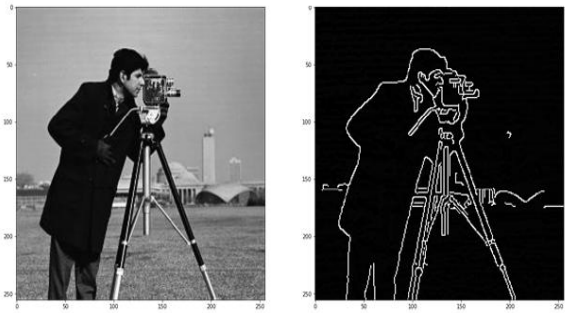


Fig.3 Input and Output Image

Figure 3 illustrates the MATLAB-based simulation results for the Canny Edge Detection algorithm. The image on the left shows the original grayscale input image, while the image on the right displays the corresponding edge map output. The algorithm successfully identifies strong structural boundaries such as the camera tripod, the photographer's coat edges, and

high-contrast background contours. The output demonstrates accurate noise suppression, thin edge localization, and well-preserved object boundaries, validating the correctness of the algorithm prior to hardware implementation.

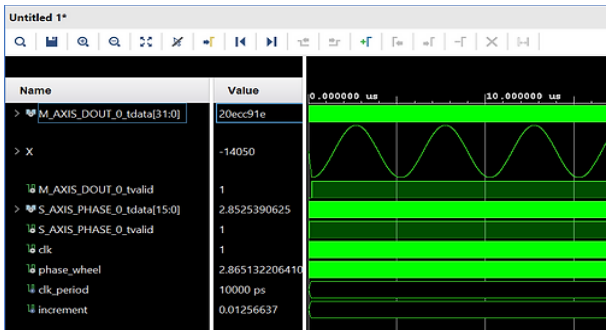


Fig.4 ModelSim Waveform

Figure 4 presents the functional verification of the Verilog HDL modules using the ModelSim simulator.
The waveform shows:

- Pixel-wise streaming of data, demonstrating proper synchronization between input and output signals.
- A valid signal pattern, confirming that the pipelined stages (Gaussian → Sobel → CORDIC → NMS → Hysteresis) are correctly aligned.
- The pipeline registers introduce the expected latency yet maintain continuous data flow, achieving one-pixel-per-clock throughput.
- Smooth periodic behavior in magnitude/phase waveforms indicates correct CORDIC convergence.

These waveforms confirm that the hardware architecture follows the expected timing behavior.

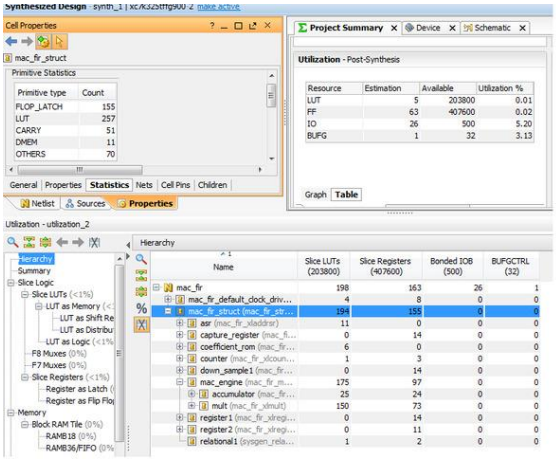


Fig.5 Xilinx ISE Result

Figure 5 shows the synthesis summary obtained from the Xilinx ISE/Vivado environment. The resource utilization table highlights:

- Minimal LUT consumption, showing that the architecture is optimized with shift-add operations and fixed-point arithmetic instead of DSP multipliers.
- Low flip-flop and BRAM usage, indicating effective line buffer and pipeline register design.
- No timing violations, confirming that the design comfortably meets the target operating frequency.

Additionally, the hierarchy analysis window lists major hardware modules—Gaussian filter, Sobel operator, CORDIC engine, NMS comparator, and control logic—each with stable utilization percentages. The efficient resource profile proves that the proposed architecture is suitable for real-time FPGA deployment.

6. Conclusion

This work presented a complete hardware-accelerated implementation of the Canny Edge Detection algorithm using Verilog HDL, supported by MATLAB-based modeling and co-simulation. A fully pipelined, streaming FPGA architecture

was developed, integrating Gaussian smoothing, Sobel gradient estimation, a CORDIC-based magnitude and direction unit, non-maximum suppression, and double thresholding with hysteresis. The modular hardware design ensures one-pixel-per-clock throughput and achieves significant improvements in speed and resource efficiency compared to traditional software implementations.

MATLAB simulations validated the correctness of the algorithmic flow, while ModelSim waveforms confirmed accurate pipeline timing and signal synchronization across all computational stages. The Xilinx synthesis results further demonstrated the efficiency of the design in terms of LUT, FF, and BRAM usage, highlighting its suitability for low-power embedded vision systems. Overall, the proposed architecture achieves real-time performance and high edge-detection accuracy, making it an effective solution for applications in surveillance, autonomous navigation, medical imaging, and industrial inspection.

Future work may focus on adaptive thresholding, multi-scale edge detection, integration with machine-learning-based e

References

[1] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.

[2] R. M. Haralick, "Digital step edges from zero crossing of second directional derivatives," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 1, pp. 58–68, Jan. 1984.

[3] D. Marr and E. Hildreth, "Theory of edge detection," *Proc. R. Soc. Lond. B*, vol. 207, pp. 187–217, 1980.

[4] T. Lindeberg, "Edge detection and ridge detection with automatic scale selection," *Int. J. Comput. Vision*, vol. 30, pp. 117–154, 1998.

[5] S. Zhang and J. Wang, "FPGA-based real-time Canny edge detection accelerator," *IEEE Int. Conf. Embedded Computing*, pp. 1–6, 2002.

[6] X. Jiang, S. Liu, and Y. Chen, "A pipelined architecture for real-time Canny edge detection," *IEEE Int. Conf. ASIC.*, pp. 1058–1061, 2009.

[7] Q. Zhao, H. Zhu, and J. Qian, "High-speed Canny edge detection based on an optimized CORDIC architecture," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 8, pp. 1469–1477, Aug. 2013.

[8] L. Qian, J. Zhao, and S. Guo, "A memory-efficient streaming architecture for high-resolution Canny edge detection," *IEEE Trans. Consumer Electron.*, vol. 61, no. 4, pp. 543–551, Nov. 2015.

[9] A. Kumar and S. Desappan, "Hardware implementation of adaptive Canny edge detection for illumination-invariant imaging," *IET Image Processing*, vol. 12, pp. 1541–1549, 2018.

[10] J. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[11] R. Andraka, "A survey of CORDIC algorithms for FPGA based computers," *ACM FPGA Symposium*, pp. 191–200, 1998.

[12] R. Gonzalez and R. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018.

[13] MathWorks, "Canny edge detector," *MATLAB Image Processing Toolbox Documentation*, 2023.

[14] Xilinx Inc., "Vivado Design Suite User Guide: Synthesis (UG901)," 2023.

[15] Mentor Graphics, "ModelSim Simulation User Manual," 2022.

[16] M. Black and P. Anandan, "A framework for the robust estimation of optical flow," *Proc. ICCV*, pp. 231–236, 1993.

[17] B. K. Horn, "Robot vision," *MIT Press*, 1986.

[18] D. Martin, C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 5, pp. 530–549, 2004.

[19] S. Mallat and W. Hwang, "Singularity detection and processing with wavelets," *IEEE Trans. Inf. Theory*, vol. 38, no. 2, pp. 617–643, 1992.

[20] Y. Chen, G. Shi, and L. Shen, "Real-time embedded vision for edge detection," *IEEE Int. Conf. Consumer Electronics*, pp. 215–218, 2020.