**RESEARCH PAPER**

# Design And Implementation Of Sobel Edge Detection Algorithm For Image Processing Applications On FPGA And ASIC

**Dr D VIJAYALAKSHMI***

*Associate professor Department of Electronics and communication Engineering
Bangalore institute of technology,Bangalore, lakshmi@bit-bangalore.edu.in

***Abstract*—** This paper investigates the implementation and comparative analysis of the Sobel edge- detection algorithm on Field-Programmable Gate Array (FPGA) and Application-Specific Integrated Circuit (ASIC) platforms. The Sobel algorithm, widely recognized in image processing and computer vision, is used to detect edges in images by calculating the gradient of pixel intensity. Its computational efficiency and significance in applications such as object detection, medical imaging, and autonomous systems make it a relevant choice for hardware acceleration.

The FPGA implementation employs the Xilinx Virtex-2 Pro, designed and simulated using Xilinx ISE tools. On the other hand, the ASIC design flow is carried out using Cadence tools, allowing for a highly optimized, application-specific approach. The paper aims to provide a comprehensive comparison of the two platforms by analyzing critical performance metrics, including timing, power consumption, area utilization, flexibility, and scalability. The study emphasizes the inherent trade-offs between the reconfigurable nature of FPGAs and the performance-oriented custom design of ASICs.

The primary purpose of this work is to guide designers and researchers in choosing the most suitable hardware platform based on specific application requirements. FPGAs are well-suited for rapid prototyping and applications requiring adaptability, while ASICs offer unmatched performance, energy efficiency, and cost-effectiveness for large-scale production. By addressing the challenges and strengths of both platforms, this Research contributes to the broader understanding of hardware acceleration for image processing tasks. The project also sets the foundation for further exploration in areas such as integrating artificial intelligence with edge-detection algorithms, developing energy-efficient hardware, and leveraging advanced FPGA and ASIC technologies for image processing applications.

Keywords— Sobel edge detection, FPGA, ASIC, Image processing, Hardware acceleration, Xilinx Virtex-2 Pro, Cadence tools, Power consumption, Area utilization, Timing analysis, Reconfigurable computing, Application-Specific Integrated Circuit, Field-Programmable Gate Array, Computer vision, Autonomous systems, Medical imaging, Performance comparison, Hardware optimization, Artificial intelligence integration, Energy-efficient hardware

**How to cite this article:** Vijayalakshmi D. Design and Implementation of Sobel Edge Detection Algorithm for Image Processing Applications on FPGA and ASIC. Int J Drug Deliv Technol. 2026;16(20s): 826-840. DOI: 10.25258/ijddt.16.20s.82

.

## I. INTRODUCTION

### *A. EDGE DETECTION*

Edge detection is a critical process in image analysis and computer vision, aimed at identifying points in a digital image where brightness changes abruptly. These points, referred to as edges, often correspond to the boundaries of objects, textures, or significant features in an image. As such, edge detection serves as a foundational step in various image processing tasks, particularly in machine vision, object recognition, and autonomous systems.

Edges play a pivotal role because they encapsulate crucial structural information about objects within an image. Despite its significance, achieving effective edge detection across diverse images remains a challenge. No single algorithm has been universally successful, and defining a specific quantitative measure of quality for edge detection continues to be an open problem in the field.

Conventional edge detection algorithms operate by examining image pixels for sudden intensity changes, typically by analyzing the gradient of pixel values. These methods identify points of maximum gradient magnitude, which correspond to edges. Classical edge detection techniques include:

Roberts Operator: A simple and fast method that computes gradients in diagonal directions.

Prewitt Operator: Focuses on detecting horizontal and vertical edges using discrete convolution kernels.

Sobel Operator: Extends the Prewitt operator by incorporating a smoothing effect, making it more robust to noise.

Canny Edge Detector: An advanced algorithm that optimizes edge detection by combining noise reduction, gradient computation, non-maximum suppression, and edge tracking by hysteresis.

These techniques, while effective in many scenarios, often face challenges in terms of accuracy, noise sensitivity, and adaptability to varying image conditions. Consequently, ongoing research continues to refine existing methods and explore new approaches for achieving robust and precise edge detection across diverse applications.

By leveraging FPGA technology, edge detection algorithms can benefit from the high parallelism and computational throughput offered by FPGAs, enabling real-time processing capabilities for applications

*Author for Correspondence: lakshmi@bit-bangalore.edu.in

requiring immediate responses, such as autonomous driving, robotics, and medical diagnostics.Field Programmable Gate Array(FPGA)

Field Programmable Gate Array (FPGA) technology has emerged as a viable alternative for implementing software algorithms in hardware, offering distinct advantages in performance, scalability, and flexibility. The inherent architectural characteristics of FPGAs make them suitable for a wide range of applications, including video surveillance, medical imaging, industrial automation, and real-time signal processing.

FPGAs are large-scale integrated circuits that can be reprogrammed to perform a variety of computational tasks. The term "field-programmable" refers to the ability of these devices to have their functionality altered post-manufacturing, while "gate array" signifies the internal architecture enabling this reprogramming capability. This dynamic adaptability allows for iterative design improvements and adjustments based on specific application needs.

The use of FPGAs in real-time image processing is particularly impactful due to their parallel processing capabilities and high computational density. Unlike general- purpose microprocessors, which rely on sequential execution, FPGAs can perform multiple computations simultaneously, significantly improving processing speed and efficiency. This advantage is further enhanced by the reprogrammable nature of FPGAs, providing a flexible platform for developing and optimizing image processing algorithms. Over the years, FPGAs have become the dominant form of programmable logic, underscoring their growing importance in modern computational frameworks.ASIC

An Application-Specific Integrated Circuit (ASIC) is a specialized semiconductor chip designed and optimized to perform a specific function or set of tasks with the highest efficiency. Unlike general-purpose processors like CPUs or GPUs, which handle diverse applications, ASICs are built for targeted tasks, offering significant benefits in performance, power consumption, and space optimization. These chips are prevalent in industries such as consumer electronics (smartphones, smart TVs), networking (routers, modems), automotive systems (engine control units, advanced driver-assistance systems), medical devices (pacemakers, imaging systems), and cryptocurrency mining (e.g., Bitcoin mining ASICs like Antminer).

ASICs are classified into three main types: full-custom, semi-custom, and structured ASICs. A full-custom ASIC is entirely designed from scratch, offering maximum performance and power efficiency but requiring significant time, cost, and expertise. Semi-custom ASICs, the most common type, use pre-designed logic blocks (standard cells) for faster design cycles and lower costs.

The ASIC design flow involves multiple steps, beginning with specification to define the desired functionality and performance. The RTL (Register Transfer Level) design stage uses hardware description languages like Verilog or VHDL to describe the chip's behavior. This is followed by logic synthesis, where the RTL code is converted into a gate-level netlist. Design verification ensures the design meets functional and timing specifications, while physical design involves critical steps like floor planning, placement, routing, and clock tree synthesis (CTS) to optimize chip layout and connectivity. Next, static timing analysis (STA) ensures the design meets timing constraints before tape-out, where the final design is sent to a foundry for fabrication using semiconductor manufacturing processes such as CMOS technology.

While ASICs require substantial upfront investment, long development cycles, and skilled engineering, they provide unparalleled benefits for high-volume production, where their efficiency outweighs initial costs. Key advantages include superior performance, reduced power consumption, compact size, and lower cost-per-unit at scale. However, their inflexibility (non-reprogrammable nature) and high costs make them less suitable for low- volume or rapidly evolving applications, where alternatives like FPGAs are preferred.

*B. Comparison of different operators*

Edge detection operators are fundamental tools in image processing for identifying boundaries or edges in an image. The major operators differ in their masks, computational efficiency, noise robustness, accuracy, and ability to detect edges in varying directions.

Roberts and Prewitt are the simplest and easiest edge detection operators to implement but lack noise suppression, while Sobel provides a good trade-off between simplicity and edge quality. For noise robustness, LoG and Canny are highly effective due to the use of Gaussian smoothing. Canny stands out as the most accurate operator, offering excellent edge localization and strong edges. In terms of computational complexity, Roberts and Prewitt are the least expensive, whereas Canny is the most computationally intensive, followed by LoG. For hardware implementation, Sobel is preferred, especially for FPGA or ASIC systems, as it balances simplicity, computational efficiency, and edge quality.

*C. Sobel Operator*

The Sobel Operator is a widely used edge detection technique in image processing and computer vision. It operates as a discrete differentiation operator, approximating the gradient of an image's intensity function. By emphasizing regions where intensity changes significantly, the Sobel operator effectively highlights edges in an image.

The Sobel operator calculates gradients in both horizontal (Gx) and vertical (Gy) directions using two distinct 3×3 convolution kernels. These kernels are specifically designed to respond maximally to edges aligned with the horizontal and vertical axes relative to the pixel grid. The application of these kernels involves convolving them with the image, yielding gradient components in the x and y directions.

The output of the Sobel operator consists of the gradient magnitude, which measures the strength of the edges, and the gradient direction, indicating the orientation of the edges. The gradient magnitude is

derived by combining the computed Gx and Gy values, providing a comprehensive representation of edge information across the image.

| **Parameters** | **Roberts** | **Prewitt** | **Sobel** | **Laplacian of Gaussian** | **Canny** |
|---|---|---|---|---|---|
| Operator type | Gradient-based | Gradient-based | Gradient-based | Second-order derivative-based | Multi-step hybrid |
| Image Grid | 2*2 | 3*3 | 3*3 | 5*5 | Multi-step with Gaussian mask |
| Edge Direction | Diagonal 45º,135º | Horizontal and Vertical | Horizontal and Vertical | All directions | All directions |
| Noise Sensitivity | High | Moderate | Moderate | Low (with smoothing) | Low (best among all) |
| Computational Complexity | Very Low | Low | Low | High | High |
| Accuracyof Detection | Low (approximate edges) | Moderate | High (smooth edges) | High | Very High |
| Smoothing Effect | None | None | Moderate (due to kernel design) | Strong (Gaussian smoothing) | Strong (Gaussian smoothing) |
| Edge Thickness | Thick | Thick | Thin to Moderate | Thin | Thin |
| Edge localization | Poor | Moderate | Good | Good | Excellent |
| Strength of Detected Edges | Weak | Moderate | Strong | Strong | Strong |

**Table 1: COMPARISON OF DIFFERENT OPERATORS**

## II. LITERATURE SURVEY

The development of Sobel edge detection algorithms has evolved significantly, from basic implementations to more complex and efficient designs suitable for real-time image processing tasks.

Gayathri A. G. and Remya Ajai A. S., "VLSI Implementation of Improved Sobel Edge Detection Algorithm," International Conference on VLSI Design and Embedded Systems, 2023, initially focused on improving the Sobel edge detection algorithm through

VLSI design techniques. This approach aimed to enhance the accuracy of edge detection while minimizing hardware complexity, making the algorithm more suitable for embedded systems and real-time applications. The key challenge was to maintain performance in demanding tasks while optimizing the use of hardware resources. The VLSI-based implementation provided a hardware-accelerated solution that could handle more intensive image processing tasks with increased efficiency [1].

Sun Jingcheng, Wang Zhengyan, and Li Zenggang, "Implementation of Sobel Edge Detection Algorithm and VGA Display Based on FPGA," Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL), 2022, pp. 321-325, discussed how the growing need for real-time image visualization led to the integration of the Sobel edge detection algorithm with FPGA-based systems. The combination of Sobel edge detection with VGA display output allowed for immediate visualization of processed images. This integration was especially beneficial for applications like video monitoring and surveillance systems, where fast edge detection and real-time output are crucial. The FPGA implementation enabled parallel processing, improving throughput and reducing latency, ensuring that edge detection was executed quickly enough to allow live display of results. [2].

Zou Xiangxi, Zhang Yonghui, Zhang Shuaiyan, and Zhang Jian, "FPGA Implementation of Edge Detection for Sobel Operator in Eight Directions," Journal of Signal Processing and System Design, 2021, pp. 200-205, introduced a more advanced version of the Sobel operator, extending its capabilities to detect edges in eight directions rather. This extension allowed the algorithm to capture more comprehensive edge information, Sobel methods might miss subtle edges. The FPGA-based implementation enabled the parallel computation of gradients across multiple directions, [3].

Abdelkader Ben Amara, Edwige Pissaloux, and Mohamed Atri, "Sobel Edge Detection System Design and Integration on an FPGA-Based HD Video Streaming Architecture," IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 7, 2020, pp. 1345-1352, detailed the next stage in development, which involved the integration of Sobel edge detection into high-definition video streaming architectures. The FPGA-based system enabled concurrent pixel processing, which sped up edge detection without compromising on quality. This real-time processing capability was essential for handling the large data throughput of HD video while maintaining the accuracy of edge detection in dynamic environments.[4].

Girish Chaple and R. D. Daruwala, "Design of Sobel Operator Based Image Edge Detection Algorithm on FPGA," International Journal of Computer Applications in Engineering Sciences, 2020, vol. 7, no. 5, pp. 115-120, focused on optimizing the Sobel operator for even higher performance by utilizing FPGA's inherent parallelism. The implementation aimed to process images in real-time by leveraging the power of FPGA architecture to accelerate edge detection tasks. This approach offered a more efficient hardware-based solution than traditional software methods, which were often too slow for real-time applications. By exploiting the parallel nature of FPGA design, the Sobel edge detection algorithm became capable of delivering fast and accurate results, making it ideal for embedded systems and other high-performance applications.[5].

Vanishree and K. V. Ramana Reddy, "Implementation of Pipelined Sobel Edge Detection Algorithm on FPGA for High-Speed Applications," International Journal of Electronics and Communication Engineering, 2021, vol. 15, no. 3, pp. 78-83, introduced the development of a pipelined Sobel edge detection algorithm, which elevated the performance of FPGA-based systems to the next level. This made it particularly suitable for high-speed applications, such as video processing, industrial automation, and autonomous systems. The pipelined architecture ensured that real-time image analysis could be performed at high speeds, meeting the demanding requirements of modern applications.[6].

This progression reflects the steady improvement of Sobel edge detection techniques, from basic VLSI implementations to advanced FPGA-based systems capable of real-time, high- performance processing in a variety of applications

## III. METHODOLOGY

The Sobel module is a critical component of the system responsible for performing edge detection on the video frames received from the camera module. It processes each frame pixel-by-pixel, applying the Sobel operator to identify edges by detecting intensity changes in the image. This module works in real time, ensuring that each frame is processed and output to the display without noticeable delay.

### *A. Key Components of the Sobel Module*

The Input Data Buffer serves as a temporary storage unit for pixel data received from the camera module. It organizes this data into a 3x3 window, allowing simultaneous access to the current pixel and its eight neighboring pixels. This structure is critical for convolution operations, ensuring that the required pixel data is readily available for edge detection.

The Convolution Kernels utilize two fixed 3x3 matrices to compute gradients in horizontal (Gx) and vertical (Gy) directions. These kernels are specifically designed to respond maximally to edges running perpendicular to their orientation. The horizontal kernel identifies changes in pixel intensity along the x-axis, while the vertical kernel detects changes along the y-axis.

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

Gx

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

Gy

**Figure1: Convolution kernels**

The Gradient Computation Unit performs element-wise multiplication and summation between the convolution kernels and the 3x3 pixel window. This process results in the computation of horizontal (Gx) and vertical (Gy) gradients. To determine the gradient magnitude, the formula |G|=|Gx|+|Gy| is used. This approximation simplifies hardware implementation by avoiding the computationally intensive square root operation while effectively highlighting edges.

The Thresholding Unit applies a predefined threshold to the gradient magnitude. This step suppresses low-magnitude gradients that are unlikely to represent significant edges, thereby enhancing the clarity of detected edges. Only high-magnitude gradients exceeding the threshold are retained and marked as edge pixels, reducing noise and irrelevant details in the output.

Finally, the Output Formatter converts the processed grayscale edge data into a display-compatible RGB format for visualization. Edges are typically represented in white (255, 255, 255), while non-edge regions are displayed in black (0, 0, 0). This formatting ensures that the edges detected by the system are visually distinct and easily interpretable.

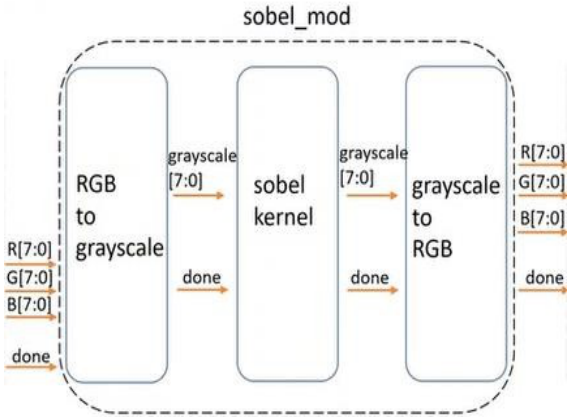



**Figure2: Sobel Module Block Diagram**

*B. RGB to GRAYSCALE*

The process of converting an RGB image to grayscale involves reducing the three color channels (Red, Green, and Blue) into a single intensity value for each pixel. This is accomplished by applying a weighted sum to the color channels, reflecting the human eye's varying sensitivity to different wavelengths of light. The resulting grayscale image effectively captures luminance, providing a simplified representation of the original image while preserving its essential details.

The conversion is mathematically expressed by the formula:

Gray = 0.299·R+0.587·G+0.114·B

RRR, GGG, and BBB are the intensity values of the red, green, and blue channels (typically 8-bit values ranging from 0 to 255). The weights 0.299, 0.587, and 0.114 reflect the relative perceptual luminance of these colors.

Y=0.3*R + 0.59*G +0.11*B

Here, RRR, GGG, and BBB represent the intensity values of the red, green, and blue channels, respectively. These are typically 8-bit values ranging from 0 to 255. The coefficients 0.2990.2990.299, 0.5870.5870.587, and 0.1140.1140.114 correspond to the relative perceptual luminance of each color channel, as the human eye is most sensitive to green, followed by red, and then blue.

For efficient hardware implementation, floating-point multiplications are avoided due to their complexity and computational cost. Instead, the weights are scaled by a factor of 1024, a power of two that simplifies arithmetic operations in digital systems. The equation is then adjusted as follows:

1024*0.3 = 307.2

1024*0.59 = 604.16

1024*0.11 =112.64

Y= (307*R + 604*G +113*B) >> 10

Here, the weights 307307307, 604604604, and 113113113 are the scaled equivalents of 0.2990.2990.299, 0.5870.5870.587, and 0.1140.1140.114, respectively. The result is normalized back to an 8-bit grayscale value by a right-shift operation of 10 bits.

*C. Read BMP File*

The process of handling BMP files, including reading, processing, and writing, requires a thorough understanding of the BMP file structure and careful

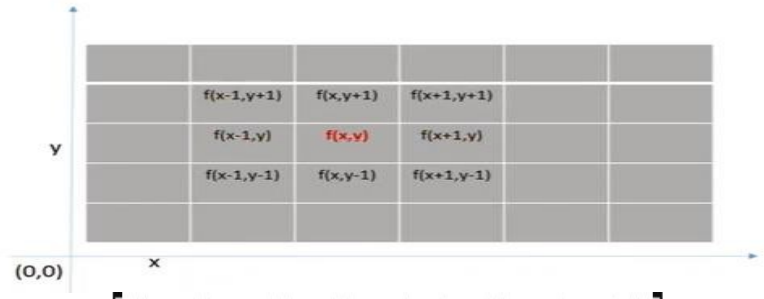
manipulation of its data. BMP files are composed of three main parts: the header, the DIB header, and the pixel data. The header contains essential metadata such as file type, size, and the offset where pixel data begins. The DIB header provides details about the image's dimensions, bit depth, and compression type. Finally, the pixel data section holds the RGB values for each pixel, stored row by row, with padding added to ensure each row's length is a multiple of 4 bytes.

To begin processing a BMP file, the file is first opened using a binary reader to load its contents into memory. The BMP and DIB headers are parsed to extract critical metadata, such as the image's width, height, and the location of the pixel data. Once this information is retrieved, the pixel data is accessed, read row by row, and any padding bytes are handled to preserve alignment. This step ensures the image's structural integrity is maintained throughout the process.

*D. Sobel Principle*

The Sobel edge detection technique is a fundamental method for identifying edges in an image by calculating the gradient of intensity at each pixel.

Each pixel in the image has neighbouring pixels that are considered when performing convolution for edge detection. For a pixel at position (x, y) its neighbouring pixels include (x−1,y−1),(x−1,y),(x−1,y+1)(x-1, y-1), (x-1, y), (x-1, y+1)(x−1,y−1),(x−1,y),(x−1,y+1), and similarly for other directions. These neighbouring pixels form a 3x3 kernel around the central pixel.




$$\begin{bmatrix} f(x-1,y-1) & f(x-1,y) & f(x-1,y+1) \\ f(x,y-1) & f(x,y) & f(x,y+1) \\ f(x+1,y-1) & f(x+1,y) & f(x+1,y+1) \end{bmatrix}$$

**Figure3: Neighbouring Pixel**

Sobel operators are two 3x3 convolution kernels that are used to compute gradients in both horizontal (Gx) and vertical (Gy) directions. These kernels emphasize edges in the respective directions and are applied to the image through convolution.

Convolution with Sobel Operators: Convolution involves sliding the Sobel kernel across the image, aligning the center of the kernel with the target pixel (x,y). Each kernel value is multiplied by the corresponding pixel value, and the results are summed to compute the gradient value for that pixel.

Once the gradients Gx and Gy are computed for each pixel, the gradient magnitude at pixel (x,y) can be derived using the formula:

$$|G(x,y)| = \sqrt{G_x(x,y)^2 + G_y(x,y)^2}$$

For hardware or computational simplicity, an approximate magnitude is often used:

$$|G(x,y)| \approx |G_x(x,y)| + |G_y(x,y)|$$

For computational simplicity in hardware, an approximate magnitude is often used, which simplifies the calculation while maintaining adequate edge detection.

Edge Identification:Significant changes in pixel intensity, indicated by high gradient values, represent edges. To identify these edges, a thresholding step is applied. Gradient magnitudes are compared to a predefined threshold, and pixels with values above this threshold are classified as edge pixels.

*E. Handling Edge Pixels*

Handling Edge Pixels in Image Processing: Zero Padding: In image processing, especially during edge detection operations like Sobel, handling the edge pixels can be a challenge. This is because the convolution process requires considering neighbouring pixels around each target pixel, and edge pixels do not have neighbouring pixels beyond the image boundary. To overcome this, padding techniques are often used.

Adding Black Pixels Around the Image (Zero Padding): One common approach to handling edge pixels is by adding black pixels, also known as zero padding, around the image. In this method, pixels with an intensity value of 0 are artificially added to the borders of the image to simulate a complete 3x3 neighborhood for each edge pixel. This padding allows the Sobel operator to still function even at the

boundaries, as the edge pixels will now have surrounding neighbors, albeit with zero values.

Advantages of Zero Padding:The main advantage of zero padding is its straightforward implementation. Since it simply involves adding a border of black pixels, it is easy to apply in convolution-based methods like Sobel edge detection. Zero padding ensures that every pixel, including those on the edges, can participate in the convolution process without modifying the image's original dimensions.

Disadvantages of Zero Padding: While zero padding is a simple and commonly used technique, it introduces certain inaccuracies at the boundaries of the image. Since the black pixels are not part of the original image data, they do not represent any real information about the image's content. As a result, the edge pixel values derived from these padded black pixels may be inaccurate, as they are influenced by artificial data rather than actual neighboring pixel values. This can lead to imperfect edge detection along the image borders, where edges may appear less defined or distorted.

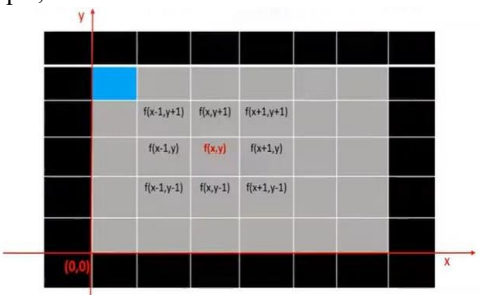



Fig4:Handling Edge Pixels

*F. FIFO Buffer*

The FIFO (First In, First Out) Buffer Mechanism is a fundamental technique used in real-time image processing systems, particularly for edge detection algorithms like Sobel. This approach ensures that pixel data is processed efficiently by storing and managing incoming pixel values in a buffer, allowing for continuous processing without requiring extensive memory storage. By utilizing a FIFO buffer system, the system can efficiently handle the processing of image pixels row by row, ensuring the creation of a 3x3 neighborhood of pixels required for operations like Sobel edge detection.

Pixel Data Storage and Buffering: In the FIFO buffer mechanism, pixel data is stored in two buffers, typically referred to as FIFO 1 and FIFO 2. FIFO 1 holds the previous row of pixel data, while FIFO 2 holds the current row. This organization allows the system to form the necessary 3x3 neighborhood for the Sobel operator by combining pixels from the two FIFO buffers and the current input stream. The shift operation in the FIFO buffers ensures that as new pixel values come in, the oldest pixels are discarded, maintaining an efficient data flow for processing.

Neighborhood Formation and Sliding Window Update: The process of edge detection, particularly the Sobel algorithm, relies on a 3x3 neighborhood of pixels to compute the gradient at each pixel. The two FIFO buffers hold the top two rows, while the current row of incoming pixels provides the third row for the neighborhood. As new pixels enter the system, the sliding window updates by shifting the FIFO buffers, and the new pixel values are used in the Sobel calculations. This continuous update process ensures that there is no need to reload or recompute previous pixel data, which makes the mechanism more efficient and faster.

Implementation Workflow: The implementation of the FIFO buffer mechanism starts with an initialization phase, where the first two rows of the image are preloaded into the FIFO buffers. This setup ensures that as soon as new pixel data begins to stream in, the system is ready to perform Sobel edge detection without any delays. The processing phase involves updating the FIFO buffers with each new pixel, forming the 3x3 neighborhood from the current and previous rows, and applying the Sobel operator to compute the gradients. The real-time operation is achieved as the system processes the pixel data continuously, forming the necessary neighborhood and performing the edge detection calculations without interruption.

Advantages of FIFO Buffers: The use of FIFO buffers in real-time image processing systems provides several advantages. First, it reduces latency since the system only buffers a small portion of the image (typically a few rows), making it more efficient than storing the entire image. This not only reduces memory usage but also allows for faster processing. Additionally, the FIFO mechanism is highly scalable, as the buffer size can be increased to handle higher- resolution images, making it adaptable to different processing requirements. The real-time capability of this approach makes it suitable for systems that need to process images continuously and with minimal delay.

State Machine Control for Real-Time Operation: To manage the flow of pixel data and ensure seamless processing, a finite state machine (FSM) or control logic is often employed. The FSM manages various states in the image processing workflow. It starts in an idle state, waiting for the first row of pixels to be loaded into FIFO 1. Once the first row is loaded, the system enters the row fill state, where the second row of pixels is loaded into FIFO 2. The processing state

follows when both FIFO buffers are filled, and the Sobel calculations begin. Finally, the end-of-frame state processes the last row of pixels, and the system is ready to process the next frame. This structured flow ensures that Sobel calculations occur only when the necessary data is available, and the system transitions smoothly between different stages of the process.

Pixel Synchronization and Boundary Handling: Pixel synchronization is essential for real-time operation, ensuring that pixel data is processed at the correct timing. The pixel clock synchronizes the input pixel stream, allowing the system to track the row and column positions of the pixels. The row counter increments when a full row of pixels is loaded into the FIFO buffers, while the column counter tracks pixel positions within the current row. The Sobel operator is applied only when at least two rows of data are buffered, and the first two pixels from the third row are available. Additionally, for boundary pixels where a full 3x3 neighborhood is not available, zero padding or other default values are used to complete the neighborhood and allow consistent edge detection.

*G. Sobel Principle Analysis*

The Sobel kernel interface plays a crucial role in managing the interaction between data buffers, such as FIFO buffers, and the computation logic, specifically the Sobel calculator, in real-time image processing systems. This interface ensures the efficient handling of pixel data streams to form the required 3x3 neighborhood necessary for convolution while maintaining the speed and accuracy required for real-time edge detection.

The Sobel operator, which is used for edge detection, processes each pixel in the image by applying a 3x3 matrix of pixel intensities. The central pixel of the neighborhood is the one being processed, and its surrounding pixels contribute to the calculation of the gradient. The structure of the kernel involves combining the current pixel with its neighboring pixels, which requires efficient buffering mechanisms like FIFO buffers to maintain the data flow.

The FIFO double line buffer is essential in convolution-based algorithms, including Sobel edge detection, due to its ability to store two consecutive rows of pixel data. This dual-buffering structure ensures that a 3x3 sliding window of pixel data is continuously available for processing. The advantage of this approach is that it allows for real-time processing without requiring the entire image to be loaded into memory.

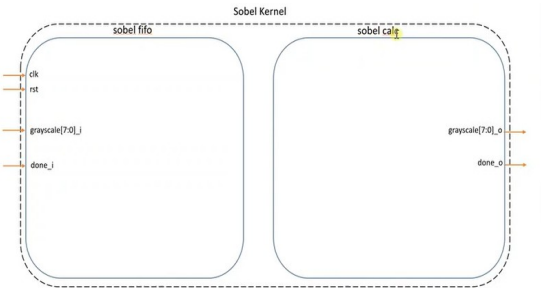
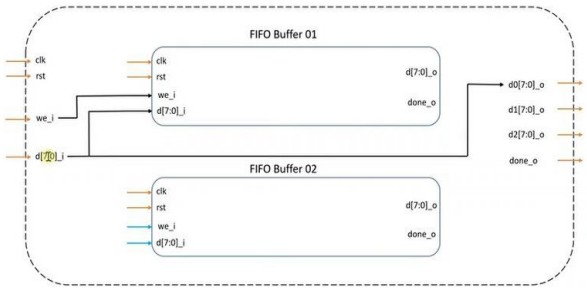



**Figure5: Sobel Kernel Analysis**

Figure




**6: FIFO Buffer**

The FIFO buffer itself is implemented using block RAM (BRAM) or shift registers in an FPGA, with each FIFO having a depth equal to the image width and a width that matches the pixel size (for example, 8 bits for grayscale images). This efficient use of on- chip memory ensures that only two rows of the image need to be buffered at a time, significantly reducing the memory requirements compared to storing the entire image. By leveraging the FIFO structure, the system is able to maintain a continuous data flow, which is necessary for real-time processing of images.

Buffers in the Sobel kernel analysis allows for efficient edge detection even in high- resolution images, making it suitable for real-time embedded systems that require fast and accurate image processing.

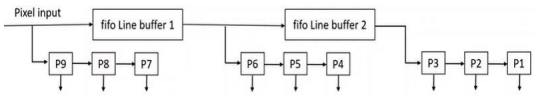



Figure7: FIFO Line Buffer

*H. Sobel Pipeline Calculation*

The Sobel pipeline architecture is designed to enhance the real-time computation of edge detection by breaking down the process into a series of sequential stages. Each stage in the pipeline performs a specific task, enabling parallel processing of different parts of the computation. This approach mimics an assembly line where multiple tasks are handled simultaneously at different stations, which in the case of image processing, allows various stages to operate on different pixels concurrently. This setup not only speeds up the overall process but also improves throughput by ensuring that multiple stages are actively working on different data, allowing for faster completion of the task.

In a pipelined architecture, data moves sequentially through stages, with each stage responsible for a particular part of the Sobel operator's calculation. For Sobel edge detection, these stages typically include grayscale conversion, FIFO buffering, convolution (which involves the gradient computation), gradient magnitude calculation, and finally, thresholding and edge detection. The advantage of pipelining here is that it allows for continuous, real-time processing of images or video streams, which is essential for applications like video processing where each frame must be handled in real-time. By breaking down the process into stages, pipelining allows for high throughput and low latency, enabling the system to process data more efficiently than if each task were completed sequentially.

The need for pipelining arises primarily from the desire to increase throughput and reduce latency. Without pipelining, each task would need to be completed one at a time, which could create bottlenecks and slow down the overall processing speed. Pipelining addresses this by allowing multiple tasks to be processed simultaneously, thereby significantly increasing the throughput of the system. This is particularly important for real- time applications such as video processing, where the system needs to handle continuous streams of data, often at rates such as 30 frames per second. The pipelined design ensures that new data can be processed as soon as the first stage becomes available, allowing for continuous operation without delays.

The synchronization of the pipeline is crucial to its proper operation. The pixel clock ensures that all stages are in sync with the incoming pixel stream, enabling coordinated data flow through the pipeline. The depth of the pipeline, or the number of stages, determines the latency between the first input pixel and the first output pixel. As the pipeline becomes filled, the throughput increases, allowing a new pixel to be processed with each clock cycle. This synchronization ensures that the data flows smoothly through the stages without any mismatches or delays.

Handling boundaries, particularly at the edges of the image (such as the first and last rows and columns), is another important consideration in Sobel edge detection. Since the Sobel operator requires a 3x3 neighborhood of pixels for each pixel, the edges of the image present incomplete neighborhoods. Two common approaches to addressing this issue are zero padding and boundary replication. Zero padding treats the missing neighbors as zero values, which simplifies the implementation but may introduce artifacts or inaccuracies at the image edges. On the other hand, boundary replication replicates the nearest available pixel values to form a complete neighborhood, which provides smoother results and reduces edge artifacts, though it may still introduce some inaccuracies depending on the image content.

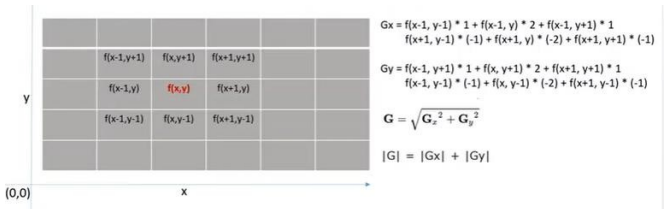
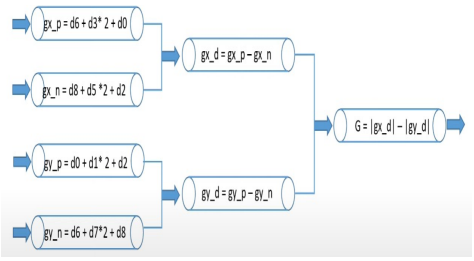



**Figure8: Sobel Pipeline Calculation**

**Figure9: Pipelining**




*I. Grayscale to RGB*

The conversion from grayscale to RGB involves transforming a grayscale image, which contains only shades of gray, into a full-color image using the RGB color model. In a grayscale image, each pixel contains a single value that represents its intensity, typically ranging from 0 (black) to 255 (white), with varying shades of gray in between. This single value in a grayscale image corresponds to the intensity of light, and there is no color information, only brightness.

The RGB color model, on the other hand, represents colors by combining different intensities of red, green, and blue light. Each pixel in an RGB image requires three values, one for each of the red, green, and blue channels. These values range from 0 to 255, where 0 indicates no intensity of that color and 255 indicates full intensity.

To convert a grayscale image to RGB, the process involves replicating the grayscale intensity value across all three channels (Red, Green, and Blue). This means that for every pixel in the grayscale image, the same intensity value is assigned to the red, green, and blue channels. For example, if a grayscale pixel has an intensity of 100, the corresponding RGB pixel will have the values (100, 100, 100). This results in a shade of gray, where all three color channels have the same intensity.

The conversion produces an RGB image that looks identical to the original grayscale image but in the format expected by systems that process RGB data. This method ensures that the original intensity of the grayscale image is preserved while making it compatible with systems that require RGB images. However, since the grayscale image does not contain any color information, the conversion does not introduce any new colors, but simply replicates the grayscale intensity into the RGB channels.

This conversion is particularly useful when working with systems or applications that require RGB images, such as video processing systems or image analysis algorithms, but where the original data is in grayscale. By replicating the grayscale value across all three RGB channels, the image retains its original grayscale appearance while being compatible with RGB-based processing systems.

## IV. IMPLEMENTATION

### *A. FPGA IMPLEMENTATION*

The implementation of the Sobel algorithm module on the Xilinx platform for the Virtex- 2 Pro hardware involves a detailed process to ensure efficient edge detection in image processing applications. The Sobel algorithm, known for its ability to highlight gradients in an image, is implemented using hardware description languages like VHDL or Verilog and simulated using Xilinx tools.

The HDL code is synthesized using Xilinx ISE tools to generate a netlist optimized for the Virtex-2 Pro FPGA. The synthesis step ensures that the module adheres to the constraints of the FPGA, such as timing, area, and resource utilization. After synthesis, the design is implemented through mapping, placement, and routing phases, ensuring optimal usage of the FPGA's resources. Simulation is performed at different levels to verify the functionality and correctness of the design. Behavioral simulation validates the logic, while post-synthesis and post-implementation simulations confirm that the design meets timing and resource constraints.

During simulation, testbench files are used to provide input image data and validate the output against expected edge-detection results. The output of the Sobel module, typically a processed image with edges highlighted, is verified against software-based Sobel implementations for accuracy. Once verified, the design is programmed onto the Virtex-2 Pro hardware using bitstream generation and FPGA configuration tools. The module can then be tested in real-time by feeding live or pre-stored image data to observe the performance and accuracy of edge detection.

The synthesis report is one of the first outputs, generated after synthesizing the HDL design. It provides details on the number of resources used, such as look-up tables (LUTs), flip-flops, block RAMs, and DSP slices, as well as the estimated maximum clock frequency. This report helps verify whether the design fits within the resource constraints of the Virtex-2 Pro FPGA and identifies any potential bottlenecks.

Timing analysis reports are generated during the implementation phase, specifically after place-and-route. These reports highlight the critical paths in the design and confirm whether the timing constraints are met. Metrics such as setup and hold time violations, clock skew, and slack are analyzed to ensure the design operates reliably at the desired clock frequency. Adjustments, such as pipelining or retiming, may be needed based on these results.

The power report provides an estimate of the dynamic and static power consumption of the module, which is particularly important for energy-sensitive applications. This report helps optimize the design by identifying components with high power consumption and

suggesting strategies to reduce power, such as clock gating or resource sharing.

Simulation results are also captured in waveform reports, showing the input and output signals over time. These waveforms are used to verify the correctness of the Sobel filter operation by comparing the processed edge-detected output with the expected results for a given input image. Error checking and debugging are conducted if discrepancies are found.

*B. ASIC IMPLEMENTATION*

The implementation of the Sobel algorithm module in the ASIC design flow using Cadence tools, specifically Genus, follows a structured process that optimizes the design for timing, area, and power. The process begins with importing the HDL code that defines the Sobel algorithm's convolution kernels and logic for edge detection. The RTL design is synthesized into a gate-level netlist, where logical optimizations and technology mapping are performed based on the target standard cell library. During this process, constraints for timing, power, and area are applied to ensure the design meets the performance requirements and remains within resource budgets.

Genus generates multiple reports during the synthesis process to provide insights into the design's performance and efficiency. The timing report identifies critical paths and slack, confirming whether the design meets timing requirements under the given constraints. The area report provides information about the total cell area used and highlights resource utilization, ensuring that the design adheres to the specified area constraints. The power report estimates both dynamic and leakage power consumption, enabling the designer to evaluate the energy efficiency of the implementation. Additionally, the utilization report offers a detailed breakdown of the logic elements, flip-flops, and memory resources used, which helps pinpoint areas for optimization.

After synthesis, the gate-level netlist is validated through simulation to verify that it maintains the functionality of the original RTL design. This netlist serves as the input for subsequent stages of the ASIC design flow, such as floorplanning, placement, and routing, performed using tools like Cadence Innovus. The reports generated during synthesis play a critical role in guiding optimizations and ensuring that the final design is efficient, manufacturable, and meets the required specifications for performance, power, and area. The Genus synthesis tool facilitates a seamless transition from RTL to a gate-level representation, providing detailed feedback and ensuring the design's readiness for physical implementation.

The implementation of the Sobel algorithm module using Cadence tools, particularly Genus, involves synthesizing the RTL design into a gate-level netlist. The synthesis process focuses on optimizing for timing, area, and power to meet the requirements of the ASIC design. Key reports are generated during this phase to evaluate the effectiveness of the optimization. The timing report highlights critical paths and slack values, ensuring that the design meets its performance goals. The area report provides insights into resource utilization, such as the number of logic cells and memory elements used, which helps maintain the design within area constraints.

The power report estimates the dynamic and leakage power consumption, helping to assess the efficiency of the implementation. These reports are crucial for making informed decisions regarding further design optimizations. After synthesis, the gate-level netlist undergoes functional verification through simulation to confirm its correctness. This netlist then serves as input for subsequent steps in the ASIC flow, including physical design tasks like floor planning, placement, and routing, ensuring that the final design is optimized for manufacturability and performance. The Genus tool, with its powerful reporting capabilities, plays a vital role in guiding the design process, ensuring that the Sobel algorithm module is efficiently implemented for the ASIC environment.

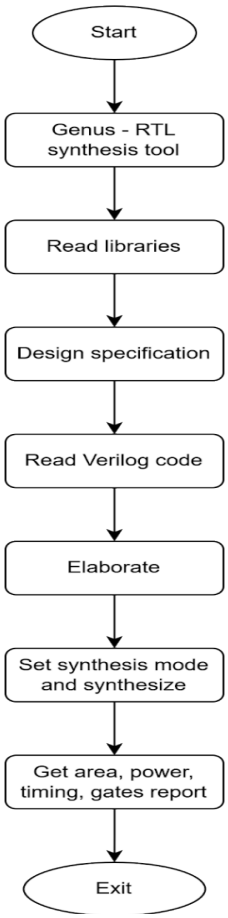



**Figure10: Flowchart for ASIC design flow**

## V. RESULTS

```
Advanced HDL Synthesis Report

Macro Statistics
# RAMs                                          : 2
 699x8-bit dual-port block RAM                  : 1
 699x8-bit dual-port distributed RAM            : 1
# Adders/Subtractors                            : 16
 10-bit adder                                   : 9
 10-bit subtractor                              : 2
 8-bit adder                                    : 5
# Counters                                      : 9
 10-bit up counter                              : 8
 8-bit up counter                               : 1
# Registers                                     : 163
 Flip-Flops                                     : 163
# Latches                                       : 9
 8-bit latch                                    : 9
# Comparators                                   : 7
 10-bit comparator greater                      : 5
 10-bit comparator less                         : 2
```

***Figure11: Advance HDL Synthesis Report***

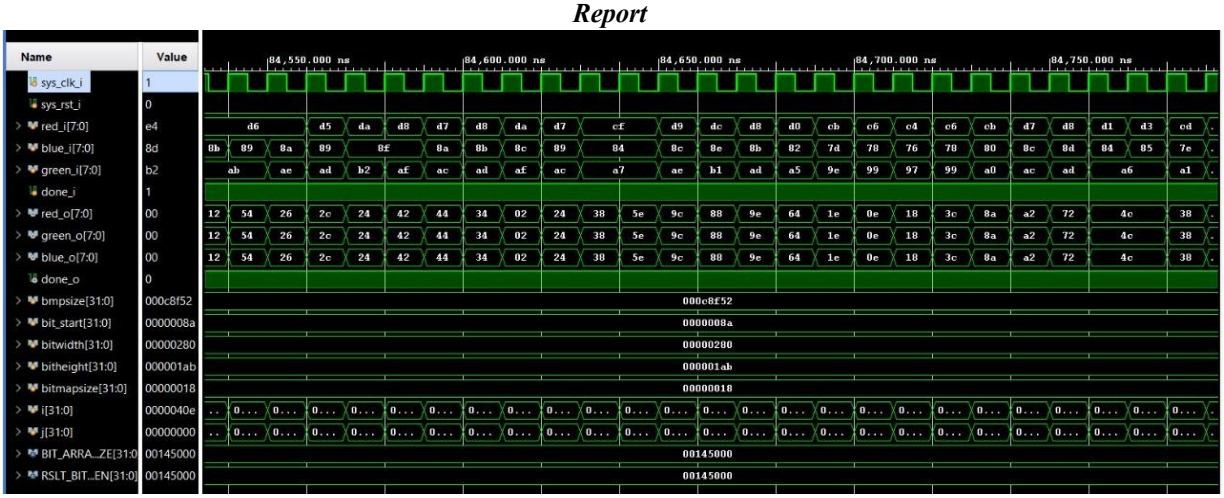



***Figure12: Simulations***

| Device Utilization Summary (estimated values) | | | |
|---|---|---|---|
| **Logic Utilization** | **Used** | **Available** | **Utilization** |
| Number of Slices | 610 | 13696 | 4% |
| Number of Slice Flip Flops | 378 | 27392 | 1% |
| Number of 4 input LUTs | 1169 | 27392 | 4% |
| Number of bonded IOBs | 45 | 556 | 8% |
| Number of BRAMs | 1 | 136 | 0% |
| Number of GCLKs | 2 | 16 | 12% |

**Figure13: Device Utilization Summary**

```
--------------------------------------------------------------------
Release 10.1 - XPower Analysis Report SoftwareVersion:K.31 (nt64)
Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.

# NOTE:  This file is designed to facilitate import into a spreadsheet program
# such as Microsoft Excel for viewing, printing and sorting.  The "|" character
# should be selected as the data field separator.

C:\isexilinix\ISE\bin\nt64\unwrapped\xpwr.exe -ise C:/sobell/sobel_algo/SObell/SObell.ise -intst
-wx sobel_top_xpwr.xml -o sobel_top.pwr

Design       | C:\sobell\sobel_algo\SObell\sobel_top.ncd                    |
Preferences  | sobel_top.pcf                                                |
Part         | 2vp30ff896-6                                                 |
Data version | ADVANCED,v1.0,05-28-03                                       |

Power summary                        |    I(mA)    |    P(mW)    |
----------------------------------------------------------------------
Total estimated power consumption    |             |       103   |
                                     ---
               Total Vccint   1.50V  |
```

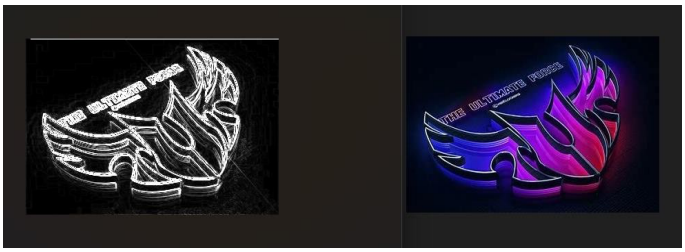


**Figure14: Power Report**

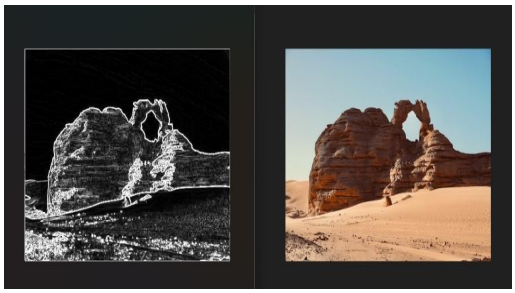


**Figure 15: Image Simulations**

The Advanced HDL Synthesis Report highlights a design that utilizes significant memory, arithmetic, and control resources. It includes two RAM instances (block and distributed), 16 adders/subtractors, 9 counters, 7 comparators, 9 latches, and 163 registers and flip-flops. The high number of arithmetic units and registers suggests a design focused on data processing with extensive pipelining or temporary storage, while the multiple counters and latches indicate complex control logic.

## *A. ASIC FLOW RESULTS*

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Dec 18 2024  01:33:40 pm
  Module:                 sobel_top
  Technology libraries:   tsmc18 1.0
                          tsmc18 1.0
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

 Instance Module  Cell Count  Cell Area  Net Area   Total Area   Wireload
-----------------------------------------------------------------------------
sobel_top              22154 928817.367     0.000   928817.367 <none> (D)
  (D) = wireload is default in technology library
```

**Figure16: ASIC Area report**

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 20.11-s111_1
  Generated on:           Dec 18 2024  01:34:32 pm
  Module:                 sobel_top
  Operating conditions:   slow (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================


Path 1: UNCONSTRAINED Setup Check with Pin kernel_inst_buff_inst_FIFO_inst_buffer_inst_1_buff_mem_reg[697][4]/CK->D
     Startpoint: (R) kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_rd_ptr_reg[0]/CK
       Endpoint: (R) kernel_inst_buff_inst_FIFO_inst_buffer_inst_1_buff_mem_reg[697][4]/D


          Setup:-   321
    Data Path:-  7955

#-------------------------------------------------------------------------------------------------------------------------------------
#                          Timing Point                           Flags   Arc   Edge   Cell     Fanout  Load  Trans Delay Arrival Instance
#                                                                                                       (fF)   (ps)  (ps)  (ps)  Location
#-------------------------------------------------------------------------------------------------------------------------------------
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_rd_ptr_reg[0]/CK       -       -     R     (arrival)  11443     -     0     0      0    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_rd_ptr_reg[0]/Q        -       CK->Q F     SDFFHQX1       5  14.8   162   331    331    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g181951/Y              -       AN->Y F     NAND2BXL       2   6.8   133   222    553    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g181945/Y              -       A->Y  R     INVX1          2  11.4   183   136    689    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g181941/Y              -       A->Y  R     AND2X4       350 1189.5  3484  2039   2728    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_drc_bufs181988/Y       -       A->Y  F     INVX1          1   6.6   402    76   2804    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_drc_bufs181986/Y       -       A->Y  R     INVX12       348 1183.2  1180   922   3726    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g179330/Y              -       A0->Y F     AOI22XL        1   3.5   506   148   3874    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g178471__5107/Y        -       B->Y  R     NAND4XL        1   3.1   247   231   4105    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g178103__7410/Y        -       B1->Y F     AOI22XL        1   3.1   587   146   4251    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g177995__5107/Y        -       D->Y  R     NAND4XL        1   3.1   252   290   4541    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g177949__5122/Y        -       B1->Y F     AOI22XL        1   3.2   269   148   4689    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g177939__5107/Y        -       C->Y  R     NAND3XL        1   3.4   203   190   4879    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g177931__9945/Y        -       C0->Y F     AOI221XL       1   3.3   267    88   4966    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_2_g177920__1705/Y        -       A0->Y R     OAI221X4     573 1057.8  3318  2267   7233    (-,-)
  hi_fo_buf186271/Y                                                    -       A->Y  R     BUFX3        128 217.6   903   722   7955    (-,-)
  kernel_inst_buff_inst_FIFO_inst_buffer_inst_1_buff_mem_reg[697][4]/D -       -     R     SDFFHQX1     128     -     -     0   7955    (-,-)
#-------------------------------------------------------------------------------------------------------------------------------------
```

**Figure 17: ASIC Timing Report**

```
Instance: /sobel_top
Power Unit: W
PDB Frames: /stim#0/frame#0
  ------------------------------------------------------------------------------
    Category         Leakage     Internal     Switching        Total     Row%
  ------------------------------------------------------------------------------
      memory     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00    0.00%
    register     2.45582e-05  1.78926e-02  9.41658e-04  1.88588e-02   88.13%
       latch     8.62392e-08  2.52468e-05  8.20491e-06  3.35380e-05    0.16%
       logic     3.41539e-06  1.40881e-03  1.09429e-03  2.50652e-03   11.71%
        bbox     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00    0.00%
       clock     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00    0.00%
         pad     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00    0.00%
          pm     0.00000e+00  0.00000e+00  0.00000e+00  0.00000e+00    0.00%
  ------------------------------------------------------------------------------
    Subtotal     2.80598e-05  1.93266e-02  2.04415e-03  2.13988e-02  100.00%
  Percentage           0.13%       90.32%        9.55%       100.00%  100.00%
  ------------------------------------------------------------------------------
```

**Figure18: ASIC Power Report**

| **Resource** | **Used** | **Available** | **Utilization (%)** |
|---|---|---|---|
| **Number of Slices** | 610 | 13,696 | 4% |

| | | | |
|---|---|---|---|
| **Number of Slice Flip-Flops** | 378 | 27,392 | 1% |
| **Number of 4-Input LUTs** | 1,169 | 27,392 | 4% |
| - Used as Logic | 464 | - | - |
| - Used as Shift Registers | 1 | - | - |
| - Used as RAM | 704 | - | - |
| **Number of Bonded IOBs** | 45 | 556 | 8% |
| **Number of BRAMs** | 1 | 136 | 0% |

Table2: FPGA RESOURCE UTLIZATION

| **PARAMETER** | **FPGA** | **ASIC** |
|---|---|---|
| Power consumption | 103mW | 213mW |
| Cells utilized | 847 | 22154 |
| Cell area available | 27392 | 928817 |
| Timing | 8588ps | 7955ps |
| Flip flops | 378 | 11213 |

Table 3: COMPARISON RESULT TABLE

## VI. CONCLUSION

In conclusion, this Research contributes a Comparative Study of Sobel Edge Detection Algorithm for Image Processing Applications: Performance Analysis on FPGA and ASIC offers a deep dive into the strengths and weaknesses of FPGA and ASIC implementations for image processing tasks. By analyzing the performance of the Sobel edge detection algorithm on two distinct platforms—FPGA (Virtex-2 Pro) and ASIC (using Cadence tools)—this work provides valuable insights into hardware selection, based on the specific requirements of an application.

The FPGA implementation, utilizing the Virtex-2 Pro device and Xilinx ISE software, stands out for its flexibility, reconfigurability, and ease of use. These features make FPGAs particularly advantageous for applications requiring rapid prototyping, iterative design, and low-volume production. The ability to modify designs during development and the availability of built-in debugging tools make FPGA a highly efficient platform for research and development, particularly in environments with evolving requirements.

The ASIC implementation, designed using Cadence tools, showcases the benefits of custom optimization. ASICs deliver significantly better performance in terms of speed, lower power consumption, and minimized silicon area, making them ideal for large-scale production runs and high-performance applications such as mobile devices, automotive systems, and data centers. Despite these advantages, ASICs require significant upfront costs, longer design cycles, and lack flexibility once fabricated, making them less suited for prototyping or applications that need frequent updates.

This comparison is essential as it highlights the trade-offs between the two technologies, helping designers and engineers make informed decisions when choosing the appropriate platform for a given application. Ultimately, this project underscores the importance of understanding the hardware and software ecosystems of both FPGA and ASIC platforms, ensuring that the design decisions align with the specific demands of the

application, whether that is high performance, low power, flexibility, or cost-effectiveness.

## REFERENCES


1. Gayathri A. G., Remya Ajai A. S., "VLSI Implementation of Improved Sobel Edge Detection Algorithm," International Conference on VLSI Design and Embedded Systems, 2023, pp. 145-150.
2. Sun Jingcheng, Wang Zhengyan, Li Zenggang, "Implementation of Sobel Edge Detection Algorithm and VGA Display Based on FPGA," Proceedings of the International Conference on Field-Programmable Logic and Applications (FPL), 2022, pp. 321-325.
3. Zou Xiangxi, Zhang Yonghui, Zhang Shuaiyan, Zhang Jian, "FPGA Implementation of Edge Detection for Sobel Operator in Eight Directions," Journal of Signal Processing and System Design, 2021, pp. 200-205.
4. Abdelkader Ben Amara, Edwige Pissaloux, Mohamed Atri, "Sobel Edge Detection System Design and Integration on an FPGA-Based HD Video Streaming Architecture," IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 7, 2020, pp. 1345-1352.
5. Girish Chaple, R. D. Daruwala, "Design of Sobel Operator Based Image Edge Detection Algorithm on FPGA," International Journal of Computer Applications in Engineering Sciences, 2020, vol. 7, no. 5, pp. 115-120.
6. Vanishree, K. V. Ramana Reddy, "Implementation of Pipelined Sobel Edge Detection Algorithm on FPGA for High-Speed Applications," International Journal of Electronics and Communication Engineering, 2021, vol. 15, no. 3, pp. 78-83.
7. V. Priyanka, Y. Sri Rama, Kadaru Sravani, Basam Kavya, “Implementation of Sobel Edge Detection with Image Processing on FPGA”, 2024 2nd World Conference on Communication & Computing (WCONF).
8. Gayathri A G, Remya Ajai A S,” VLSI Implementation of Improved Sobel Edge Detection Algorithm”, International Conference on Communication, Control and Information Sciences (ICCISc).
9. Xiangxiang Wei, Gao-Ming Du, Xiaolei Wang, Hongfang Cao, Shijie Hu, Duoli Zhang, Zhenmin Li, “FPGA Implementation of Hardware Accelerator for Real-time Video Image Edge Detection”,
   International Conference on Anti-counterfeiting, Security, and Identification (ASID).
10. V. S. Seleznev, E. O. Antonova, A. V. Iluhin, R. A. Gematudinov, L.Yu. Isaeva, “Implementation on Sobel Field-Programmable Gate Array Detector for Identification of Vehicles”, Intelligent Technologies and Electronic Devices in Vehicle and Road Transport Complex (TIRVED).
11. Fei Liu, Jipeng Wang, Bingqiang Liu, Run Min, Guoyi Yu, Fengwei An, “Low Computation and High Efficiency Sobel Edge Detector for Robot Vision”, IEEE International Conference on Real-time Computing and Robotics (RCAR).
12. R. Fisher, "Sobel Edge Detection," HIPR2, University of Edinburgh, https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm
13. "Sobel Filter," ScienceDirect, https://www.sciencedirect.com/topics/computer-science/sobel-filter
14. Xilinx, "Virtex-II Pro Platform FPGA Handbook," https://www.xilinx.com/support/documentation-navigation/silicon-devices/mature- products/virtex-ii-pro.html
15. ChipVerify, "ASIC and SoC Design Flow," https://www.chipverify.com/verilog/asic- soc-chip-design-flow