School

of

Electronics and Communication Engineering

Minor Project Report

on

Edge Detection Using FPGA-Based 2D Convolution

By:

1. **Nidhi Desai**	01FE22BEC189
2. **Devika Vijapur**	01FE22BEC197
3. **Aishwarya Naik**	01FE22BEC213
4. **Smita Ganur**	01FE22BEC247

Semester: V, 2024-2025

Under the Guidance of

Supriya K

K.L.E SOCIETY'S
KLE Technological University,
HUBBALLI-580031
2024-2025

SCHOOL OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that project entitled **"Edge Detection Using FPGA-Based 2D Convolution "** is a bonafide work carried out by the student team of **Nidhi Desai(01FE22BEC189),Devika Vijapur(01FE22BEC197),Aishwarya Naik(01F E22BEC213),Smita G(01FE22BEC247)** The project report has been approved as it satisfies the requirements with respect to the mini project work prescribed by the university curriculum for BE (V semester) in School of Electronics and Communication Engineering of KLE technological University for the academic year 2024-2025.

Supriya K	**Dr.Sunita Budihal**	**Dr.B S Anami**
Guide	**Head of School**	**Registrar**

External Viva:

Name of Examiners	**Signature with date**
1.	
2.	

ACKNOWLEDGMENT

We extend our heartfelt gratitude to Prof.Supriya K, our esteemed project guide, whose unwavering support and guidance were instrumental in the successful completion of our project titled Detection Using FPGA-Based 2D Convolution . Her dedication and timely advice were invaluable, and we owe the realization of our project to her expertise and mentorship. Despite her demanding schedule, Supriya K. generously shared her knowledge, paving the way for our project's success. We would also like to express our sincere appreciation to the teaching and non-teaching staff at the SoECE. Their constant encouragement, support, and supervision played a crucial role in our project journey.

-The project team

ABSTRACT

Edge detection is a fundamental step in many image processing and computer vision applications, enabling the identification of object boundaries and significant features within an image. This project focuses on the design and implementation of edge detection system using hardware acceleration techniques to achieve high performance and low latency. The system processes input images by converting them to grayscale, applying a 2D convolution operation with an edge detection kernel, and generating output images that highlight prominent edges. Through a hardware-software co-design approach, the computationally intensive filtering operation is offloaded to dedicated hardware, while control and data management remain in software. The design is validated through simulation and experimental testing, demonstrating a substantial improvement in processing speed compared to a purely software-based implementation. This work highlights the effectiveness of combining hardware acceleration with software flexibility to enable efficient image processing suitable for applications such as autonomous systems, surveillance, and medical imaging.

Contents

1 Introduction **8**
1.1 Motivation . . . 9
1.2 Objectives . . . 10
1.3 Literature survey . . . 10
1.4 Problem statement . . . 12
1.5 Organization of the report . . . 13

2 Preliminaries **14**
2.1 Edge detection basics . . . 14
2.2 2D Convolution Operation . . . 15
2.3 Zynq FPGA Architecture and PYNQ-Z2 Board Overview . . . 16

3 System design **19**
3.1 Functional Block Diagram . . . 19
3.2 Morphological Chart . . . 20
3.3 Design Alternatives . . . 21
3.4 Design Choices and Justifications . . . 22

4 Implementation details **24**
4.1 Edge Detection Algorithm Using 2D Convolution Implemented on PYNQ-Z2 FPGA Platform . . . 24
4.2 Flowchart . . . 27
4.3 Results . . . 29

5 Conclusions and Future Scope **31**
5.1 Conclusion . . . 31
5.2 Future Scope . . . 31
5.3 Application in the Societal Context . . . 32

List of Figures

1.1 Edge detection 12

2.1 PYNQ-Z2 board 17

3.1 Functional block diagram of real-time edge detection using FPGA 19

4.1 Logic of Sobel edge detection filter using 2D Convolution implemented in Vitis HLS 25
4.2 Performance parameters of the developed sobel logic 25
4.3 Block design in Vivado Design Suite 26
4.4 Deployment of the design onto the hardware using Jupyter Notebook . . . 26
4.5 Flowchart representing real-time edge detection implemented on software and hardware 28
4.6 Comparison of input and edge detection results 29
4.7 High-Level Synthesis (HLS) report showing hardware resource utilization for each module in the Sobel Edge Detection pipeline on PYNQ-Z2. 29

Chapter 1

Introduction

Edge detection is a critical operation in digital image processing, used to identify points in an image where the intensity changes sharply. These changes typically correspond to physical discontinuities in the scene, such as object boundaries, surface textures, or shadows. The ability to extract this structural information is essential for numerous vision-based applications that require a reliable understanding of the environment. Among these, Advanced Driver Assistance Systems (ADAS) stand out as a domain where accurate and real-time edge detection plays a pivotal role in ensuring safety, situational awareness, and automation.

In ADAS, vehicles must process visual data from on-board cameras and sensors in real time to make decisions such as lane detection, obstacle avoidance, pedestrian recognition, and traffic sign identification. Each of these tasks depends on the system's ability to quickly and precisely identify the edges of relevant objects in the vehicle's surroundings. However, traditional software-based approaches that rely on CPUs or GPUs often fail to meet the stringent latency and power efficiency requirements of in-vehicle systems, especially when dealing with high-resolution video streams or multiple camera inputs. This limitation becomes even more critical in safety-critical scenarios where delays in processing can result in life-threatening outcomes.

To address these challenges, Field-Programmable Gate Arrays (FPGAs) offer a powerful hardware-based solution for accelerating edge detection and other image processing tasks. Unlike general-purpose processors that execute instructions sequentially or in limited parallelism, FPGAs provide a fully parallel and customizable architecture that can be tailored to the specific needs of real-time systems. They allow designers to implement deeply pipelined and parallelized 2D convolution operations—the mathematical core of most edge detection algorithms—enabling significant improvements in throughput and responsiveness. Moreover, FPGAs exhibit low power consumption and deterministic performance, both of which are highly desirable in automotive systems where reliability and energy efficiency are paramount.

In this project, the focus is on implementing a edge detection system using FPGA-based 2D convolution. The 2D convolution technique involves sliding a kernel matrix over the input image to detect changes in pixel intensity that signify edges. By offloading this computation to an FPGA, the system benefits from the inherent parallelism and hardware-level optimization that the platform offers. This implementation serves as a foundation for vision-based modules within ADAS, such as lane departure warning systems and collision detection mechanisms, where every millisecond of processing time can make a critical difference.

Through the exploration of FPGA-based acceleration, this work aims to demonstrate the practical benefits of integrating custom hardware designs into intelligent transportation systems. The report discusses the design methodology, hardware implementation strategies, and performance evaluation of the system, with the broader objective of contributing toward safer and smarter automotive technologies.

1.1 Motivation

As modern systems increasingly rely on intelligent decision-making driven by visual perception, the importance of image processing has grown across a wide range of domains. One of the most vital components of this visual pipeline is edge detection—a foundational operation used to identify object boundaries, structural features, and significant transitions in an image. Accurate and timely edge detection is essential in various applications such as robotics, medical imaging, industrial automation, and most critically, in automotive safety systems like Advanced Driver Assistance Systems (ADAS).

ADAS has transformed the transportation landscape by equipping vehicles with the ability to perceive and respond to their surroundings. Functions such as lane keeping, collision avoidance, traffic sign recognition, and pedestrian detection all rely on accurate edge information extracted from video feeds. In such systems, processing delays or inaccuracies in detecting visual features can lead to safety risks. Therefore, there is a pressing need for image processing solutions that are not only accurate but also fast, power-efficient, and reliable.

Traditional processing platforms such as CPUs and GPUs, while powerful and flexible, often fall short when it comes to meeting the strict constraints of embedded systems, especially in power- and space-constrained environments like vehicles. Software-based edge detection on general-purpose processors introduces latency due to sequential execution and overhead from instruction fetching, context switching, and memory access. Moreover, power-hungry GPUs are not ideal for deployment in systems that require sustained operation under thermal and energy limitations.

This growing gap between computational demand and platform capability has motivated a shift towards hardware acceleration, with Field-Programmable Gate Arrays (FPGAs) emerging as a compelling solution. FPGAs offer a unique combination of high-performance computation, fine-grained parallelism, and reconfigurability, making them particularly suitable for implementing low-level, repetitive operations such as 2D convolution. Unlike fixed-function ASICs or software-based processors, FPGAs allow developers to tailor the architecture for a specific task, enabling optimized throughput, reduced latency, and deterministic behavior—all of which are critical for safety and performance in automotive applications.

The motivation for this project is rooted in demonstrating how FPGA-based edge detection can address the shortcomings of traditional approaches while meeting the rigorous demands of real-time embedded systems. By implementing 2D convolution directly in reconfigurable hardware, we aim to achieve faster image processing with lower power consumption and increased reliability. This hardware-based approach is particularly aligned with the vision of intelligent, autonomous mobility, where vehicles must interpret their environment continuously, react instantly, and maintain operational safety under all conditions.

Ultimately, this project is motivated not just by technological curiosity, but by a broader goal—to contribute to the development of safer, smarter, and more efficient embedded vision systems that can power the next generation of autonomous vehicles and intelligent machines.

1.2 Objectives

1. To implement edge detection using FPGA-based 2D convolution.
2. To utilize FPGA's parallelism for fast and efficient image processing.
3. To evaluate the system's performance in terms of speed and accuracy.
4. To design a low-power, reliable solution suitable for embedded automotive use.

1.3 Literature survey

Several studies have evaluated various methods for edge detection using FPGA platforms, focusing on optimizing performance, resource utilization, and real-time capability. In [14], a generalized FPGA accelerator for 2D convolution achieved a throughput exceeding 4 GOPS and supported multiple kernel sizes and strides, demonstrating its flexibility and scalability across image processing pipelines. The architecture emphasized modularity and was optimized for both low-latency execution and high parallelism, making it suitable for deployment in autonomous systems and embedded vision tasks. The work in [5] introduced a hardware-optimized version of the Canny edge detection algorithm tailored for FPGA platforms, featuring enhancements in gradient computation and non-maximum suppression. By structuring the pipeline to minimize memory bottlenecks and latency, the system significantly outperformed CPU implementations in time-critical tasks such as live surveillance and navigation systems.

The study in [9] utilized the Xilinx PYNQ-Z2 board to implement a real-time vision system leveraging the ease of Python programming alongside hardware-accelerated IP cores. The PYNQ framework facilitated a smooth interface between software-level applications and FPGA-accelerated hardware overlays, enabling rapid prototyping of image processing functions like filtering and edge detection. In [16], further advancement was made by integrating the Xilinx Deep Processing Unit (DPU) with HDMI on the PYNQ board, allowing direct image capture and AI-based inference on live video feeds. This enabled real-time image enhancement and convolutional neural network processing within a compact, power-efficient edge computing platform. In [4], low-precision YOLOv3 models were successfully deployed on PYNQ hardware, illustrating the ability to combine edge detection with real-time object detection using quantized deep learning models for optimized speed and resource utilization.

The advancement of FPGA-based edge detection outside the PYNQ ecosystem is evident in [19], where high-level synthesis was utilized to accelerate Sobel edge detection, demonstrating efficient loop pipelining and memory reuse strategies that reduced development time while maintaining high performance. Study [13] presented a Verilog-based Sobel implementation focused on energy efficiency, achieving low power consumption and high-speed processing suitable for portable or battery-operated devices. The approach validated the performance advantage of FPGAs over CPUs for repetitive convolution operations. In [10], a resource-conscious Sobel edge detector was proposed, optimizing LUT and BRAM usage for area-constrained FPGAs such as PYNQ-Z1 and Z2. This made the design especially useful for embedded applications where silicon real estate and power are limited. Collectively, these studies establish that FPGA platforms, particularly the PYNQ family, offer an excellent balance of performance, scalability, and power efficiency for real-time 2D convolution-based edge detection.

[1] explores the feasibility of implementing 2D convolution operations on FPGAs using a systolic array architecture combined with the Residue Number System (RNS). The authors propose leveraging the parallelism inherent in systolic arrays and the carry-free arithmetic of RNS to enhance computational efficiency. The design aims to optimize performance for digital signal processing tasks by reducing the complexity of arithmetic operations and improving data throughput. The study demonstrates that integrating RNS with systolic arrays can lead to significant improvements in speed and resource utilization for 2D convolution operations on FPGA platforms. The study [6] presents an energy-efficient design for performing image convolution operations on FPGA platforms. The authors focus on optimizing the energy consumption of convolution processes, which are fundamental in image processing applications. By analyzing various design parameters and their impact on energy efficiency, the paper provides insights into achieving high-performance convolution operations with reduced power usage. The proposed designs demonstrate significant improvements in energy efficiency, making them suitable for applications where power consumption is a critical concern. The paper [14] proposes an FPGA-based architecture designed to accelerate convolution operations, which are computationally intensive and prevalent in convolutional neural networks (CNNs). The authors develop an IP core capable of processing convolutional layers efficiently, targeting edge-AI applications. The design, implemented using Verilog HDL, is adaptable to various FPGA families. Experimental results indicate that the proposed solution achieves substantial performance gains, making it a viable option for real-time image processing tasks in resource-constrained environments

The paper [17] presents a high-performance, resource-efficient architecture using a systolic array for real-time image convolution. By incorporating dual-port RAM as FIFO buffers and optimizing with adder trees, the design enables parallel processing and faster throughput. Two VHDL-based designs implemented on a Spartan3-E FPGA show that the second version achieves twice the throughput with reduced silicon usage, proving the effectiveness of this approach for real-time image processing on FPGAs. The study [7] introduces a flexible and efficient FPGA-based architecture that supports real-time image convolution through dynamic partial reconfiguration. The proposed system enables parallel processing of image data and allows the convolution kernel to be updated at runtime without halting the entire system. This adaptability makes the architecture well-suited for applications requiring varying filters or real-time adaptability, demonstrating improved performance and resource optimization over traditional static implementations.

The paper [8] focuses on designing an optimized 2D convolution processor architecture tailored for FPGA platforms to enhance performance in image filtering applications. It explores techniques such as loop unrolling, pipelining, and efficient memory access to reduce latency and maximize throughput. The proposed design demonstrates significant improvements in speed and resource utilization compared to conventional implementations, making it highly effective for real-time image processing tasks where both performance and area efficiency are crucial.

The reviewed literature demonstrates significant advancements in FPGA-based 2D convolution implementations for real-time image processing applications, showcasing various optimization techniques such as parallelism, dynamic reconfiguration, and resource-efficient architectures. However, many existing solutions still face challenges related to balancing hardware resource utilization, processing speed, and flexibility for diverse image processing tasks. Our project aims to bridge this gap by developing an optimized FPGA implementation of 2D convolution that not only maximizes processing throughput but also maintains resource efficiency and adaptability. By leveraging insights from prior work and introducing novel architectural enhancements, our approach seeks to improve real-time performance while minimizing power consumption and hardware overhead, thus addressing critical needs in embedded and edge computing environments.

1.4 Problem statement

Edge detection is crucial for Advanced Driver Assistance Systems (ADAS) to accurately interpret visual data for vehicle safety and navigation. However, traditional software-based methods on CPUs or GPUs often fail to deliver low latency, and energy efficiency required in embedded automotive systems. This project aims to address these challenges by designing and implementing an edge detection system using FPGA-based 2D convolution, leveraging hardware parallelism to achieve fast, accurate, and power-efficient image processing suitable for safety-critical automotive applications.

Figure 1.1 [11] refers to an example of how edges are detected when an image is given as input.

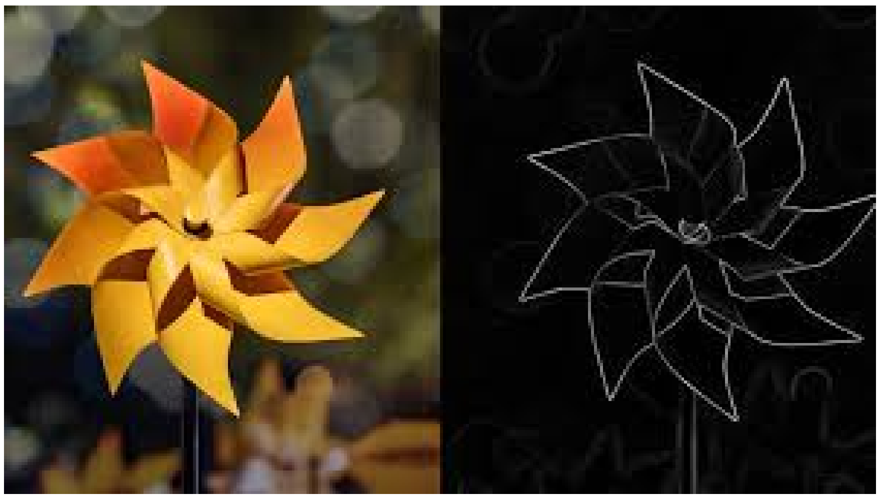


Figure 1.1: Edge detection

1.5 Organization of the report

- Chapter 2:Preliminaries - This section presents foundational concepts relevant to the project, including an overview of 2D convolution and common edge detection algorithms (like Sobel). It also covers the basics of FPGA technology, its architecture, and why it is suited for accelerating image processing tasks.
- Chapter 3: System Design- This chapter introduces the overall system architecture and the functional block diagram of the FPGA-based edge detection module. It details the design choices made for the 2D convolution implementation.
- Chapter 4: Implementation details- This section outlines the hardware specifications and development environment used for FPGA programming. It describes the step-by-step implementation of the 2D convolution algorithm on the FPGA. It also presents the flowchart of the edge detection process from image input to output.
- Chapter 5: Results and discussions: This chapter analyzes the performance metrics of the FPGA-based edge detection system, such as processing speed (frame rate), resource utilization, power consumption, and edge detection accuracy. It compares these results with software-based implementations and discusses the implications for ADAS applications.
- Chapter 6: Conclusion- The study concludes by summarizing the success of the FPGA-based approach in achieving fast, efficient, and reliable edge detection. It highlights the potential impact of this technology in enhancing the perception capabilities of Advanced Driver Assistance Systems, thereby contributing to safer and smarter vehicles.

Chapter 2

Preliminaries

This section outlines the fundamentals of edge detection and 2D convolution, which are vital for extracting structural features from images. It highlights various edge detection techniques like Sobel, Canny, and Roberts, emphasizing the Sobel operator's suitability for hardware implementation. The 2D convolution process is explained as a sliding filter mechanism for detecting intensity changes. Additionally, it describes the architecture of the PYNQ-Z2 board, which integrates ARM processing with FPGA logic for efficient image processing in applications such as ADAS.

2.1 Edge detection basics

Edge detection is a critical process in image processing used to identify significant transitions in intensity within an image, which often correspond to object boundaries or changes in surface properties. Detecting these edges helps in understanding the structure and features of the scene, which is essential in many applications like object recognition, image segmentation, medical imaging, and advanced driver assistance systems (ADAS). Several edge detection methods can be utilized, each with specific characteristics:

1. Sobel Operator: Uses two 3×3 convolution kernels to approximate the gradient of the image intensity in the horizontal and vertical directions. It emphasizes edges by calculating the magnitude of these gradients.

2. Prewitt Operator: Similar to Sobel but uses a different set of kernels to compute gradients. It is simpler but less sensitive to noise.

3. Roberts Cross Operator: Uses a small 2×2 kernel to detect edges diagonally. It is fast but more sensitive to noise and less accurate on noisy images.

4. Canny Edge Detector: A multi-stage algorithm involving Gaussian smoothing, gradient calculation, non-maximum suppression, and hysteresis thresholding. It produces accurate and thin edges with good noise reduction but is computationally intensive.

5. Laplacian of Gaussian (LoG): Combines Gaussian smoothing and the Laplacian operator to detect edges by identifying zero-crossings in the second derivative of the image.

Among these methods, the 'Sobel filter' is widely preferred for hardware implementations like FPGA-based edge detection due to its balance between simplicity, noise robustness, and effective edge highlighting. The Sobel operator uses larger kernels than Roberts, providing better noise immunity, and is computationally less complex than the Canny detector, making it suitable for real-time processing on embedded platforms.

The Sobel operator applies two 3×3 convolution kernels, G_x and G_y, to approximate the gradients in horizontal and vertical directions respectively:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \text{[12]}$$

For each pixel, the gradients in the x and y directions are calculated as:

$$S_x = G_x * I, \quad S_y = G_y * I \qquad \text{[12]}$$

where $*$ denotes the convolution operation and I is the input image.
The gradient magnitude, representing the edge strength, is then computed as:

$$S = \sqrt{S_x^2 + S_y^2} \qquad \text{[12]}$$

In practice, for faster computation, the magnitude is often approximated by:

$$S \approx |S_x| + |S_y| \qquad \text{[12]}$$

Edges are then detected by thresholding the gradient magnitude to distinguish significant edges from noise.
Overall, the Sobel operator's combination of reasonable noise resistance, computational simplicity, and effectiveness in edge highlighting makes it ideal for real-time FPGA implementation in applications like ADAS, where speed and accuracy are critical.

2.2 2D Convolution Operation

In image processing, **2D convolution** is a fundamental operation used to extract features such as edges, textures, and patterns from an image. It works by sliding a small matrix known as a *kernel* or *filter* over an image and computing the weighted sum of the overlapping pixel values. This process allows highlighting specific structures in the image, such as edges.

Mathematically, the 2D convolution of an image I with a kernel K of size $m \times n$ is defined as:

$$S(i,j) = \sum_{k=-a}^{a} \sum_{l=-b}^{b} K(k,l) \cdot I(i-k, j-l) \qquad \text{[2]}$$

Where:

- $S(i,j)$ is the output (convolved) image at position (i,j),
- $I(i,j)$ is the input image,
- $K(k,l)$ is the kernel (filter),

- $a = \left\lfloor \frac{m}{2} \right\rfloor$, $b = \left\lfloor \frac{n}{2} \right\rfloor$,
- The summation is over the dimensions of the kernel centered at pixel (i, j).

For example, if we apply the Sobel operator's G_x kernel to detect vertical edges, the convolution process involves centering the kernel on each pixel and calculating the dot product between the kernel and the corresponding 3×3 patch of the image:

$$G_x(i,j) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} G_x(k,l) \cdot I(i-k, j-l) \qquad [12]$$

Similarly, the horizontal edge component is calculated using:

$$G_y(i,j) = \sum_{k=-1}^{1} \sum_{l=-1}^{1} G_y(k,l) \cdot I(i-k, j-l) \qquad [12]$$

The gradient magnitude at each pixel is then computed as:

$$G(i,j) = \sqrt{G_x(i,j)^2 + G_y(i,j)^2} \quad \text{or} \quad G(i,j) \approx |G_x(i,j)| + |G_y(i,j)| \qquad [12]$$

This gradient magnitude image represents the **edge strength**, highlighting regions with sharp intensity changes.

Convolution is linear and shift-invariant, making it suitable for real-time image processing systems, especially when implemented on hardware like **FPGAs**, which can execute multiple convolution operations in parallel due to their reconfigurable and parallel architecture.

2.3 Zynq FPGA Architecture and PYNQ-Z2 Board Overview

The PYNQ-Z2 board is built around the Xilinx Zynq-7000 SoC, which features a powerful hybrid architecture combining a dual-core ARM Cortex-A9 Processing System (PS) with Programmable Logic (PL) on a single chip. This integration offers the flexibility of FPGA fabric alongside the processing capabilities of an embedded ARM processor, making it ideal for applications requiring both software programmability and hardware acceleration.

On the PYNQ-Z2, the Processing System runs a Linux-based environment, managing general-purpose tasks such as file handling, network communication, and user interface through Jupyter notebooks. Meanwhile, the Programmable Logic fabric handles computation-intensive operations such as real-time image processing and 2D convolution for edge detection. This hardware-software co-design approach allows developers to efficiently accelerate performance-critical functions on the FPGA while maintaining easy control and interaction via the ARM processor.

The PYNQ-Z2 board further supports dual communication interfaces including USB for serial terminal access and Ethernet for network connectivity, enabling seamless development and deployment workflows. This architecture makes the PYNQ-Z2 an excellent platform for implementing advanced driver assistance systems (ADAS) that rely on fast and reliable real-time image processing capabilities. Figure 2.1 represents the PYNQ-Z2 board [15].

PYNQ-Z2 Board Features [3]

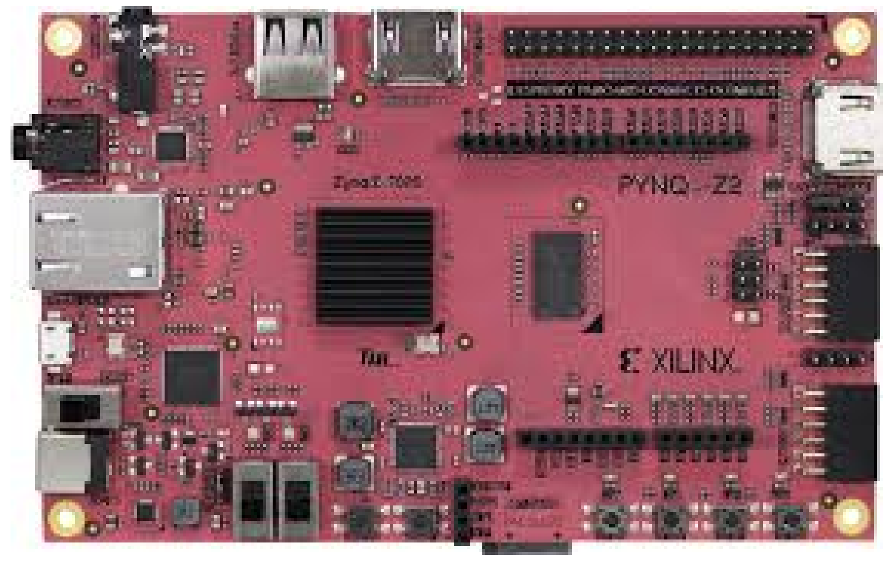


Figure 2.1: PYNQ-Z2 board

- **Zynq SoC:** Xilinx ZYNQ XC7Z020-1CLG400C
- **Processor:** 650 MHz ARM Cortex-A9 dual-core processor
- **Programmable Logic:**
  - 13,300 logic slices, each with four 6-input LUTs and 8 flip-flops
  - 630 KB block RAM
  - 220 DSP slices
  - On-chip Xilinx analog-to-digital converter (XADC)
  - Programmable via JTAG, Quad-SPI flash, and MicroSD card
- **Memory and Storage:**
  - 512 MB DDR3 with 16-bit bus @ 1050 Mbps
  - 16 MB Quad-SPI Flash with factory programmed 48-bit globally unique EUI-48/64™ compatible identifier
  - MicroSD card slot for external storage and booting
- **Power:**
  - Powered via USB or 7V to 15V external power regulator
- **USB and Ethernet Interfaces:**
  - Gigabit Ethernet PHY for high-speed networking
  - Micro USB-JTAG programming circuitry

  - Micro USB-UART bridge for serial communication
  - USB 2.0 OTG PHY (host only support)

- **Audio and Video:**
  - Two HDMI ports (input and output)
  - 24-bit I2S DAC with 3.5 mm TRRS audio jack
  - Line-in with 3.5 mm jack

- **User Interface Components:**
  - 4 push-buttons
  - 2 slide switches
  - 4 LEDs
  - 2 RGB LEDs

- **Board Dimensions:** 87 mm x 140 mm (3.43" x 5.51")

Software Environment

The board runs a Linux-based PYNQ image optimized for embedded development. This environment provides:

- A pre-configured Jupyter Notebook server accessible through a web browser for interactive development.
- Python APIs (PYNQ libraries) that simplify FPGA overlay management and hardware acceleration tasks.
- Tools to program, deploy, and debug hardware accelerators directly from Python.

Relevance to Edge Detection Project

The PYNQ-Z2's programmable logic fabric is well-suited for implementing image processing algorithms such as the Sobel filter for edge detection. Using Vitis HLS, custom hardware IPs for 2D convolution can be generated and loaded as overlays. This setup leverages the FPGA for computationally intensive tasks while using the ARM processor to handle control, data movement, and visualization, thus enabling efficient edge detection applications, which are critical for advanced driver assistance systems (ADAS).

Chapter 3

System design

This section describes the overall architecture of the edge detection system implemented on the Zynq FPGA platform. The design integrates a custom 2D convolution IP core based on the Sobel filter within the FPGA fabric to perform high-speed, parallel edge detection. The dual-core ARM Cortex-A9 processor handles system control, configuration, and communication, while the programmable logic executes the computationally intensive convolution operations. The data flow from image acquisition through processing to output is managed efficiently to meet real-time performance requirements.

The system uses the PYNQ-Z2 development board, which offers versatile interfaces like USB and Ethernet for programming and data transfer. Running Linux with Jupyter notebooks enables easy interaction, testing, and deployment of the edge detection algorithm. This combination of programmable logic and processor system allows seamless hardware-software co-design, making the solution suitable for advanced applications like ADAS, where quick and reliable edge detection is crucial for vehicle safety and environment perception.

3.1 Functional Block Diagram

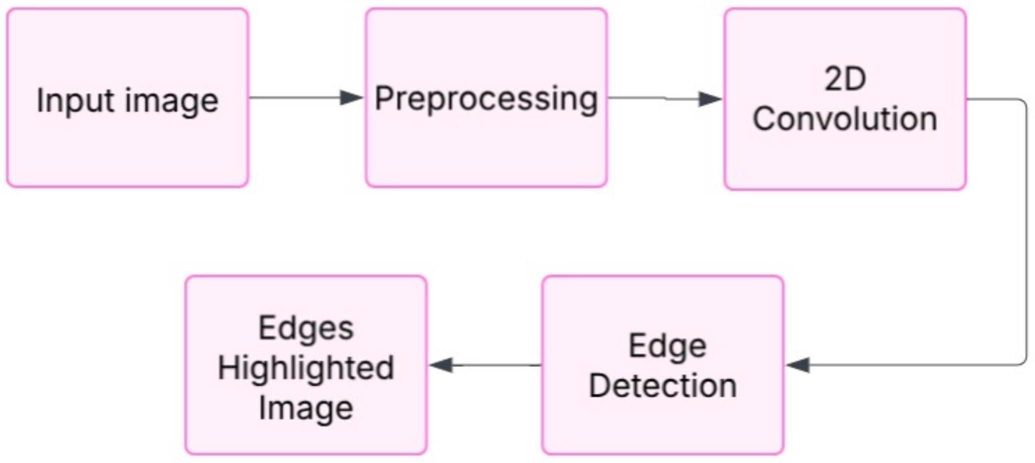


Figure 3.1: Functional block diagram of real-time edge detection using FPGA

Edge detection is a vital task in image processing, particularly in applications like autonomous systems, surveillance, and object recognition, where immediate response is crucial. Implementing edge detection on an FPGA (Field Programmable Gate Array) platform leverages its high-speed, parallel-processing capabilities. The functional block diagram is as shown in the Figure 3.1. The process begins with an input image, typically captured from a camera or stored in memory. These images are often in RGB (Red, Green, Blue) format, which contains color information. However, since edge detection primarily relies on intensity changes rather than color, the RGB image is converted to grayscale during the pre-processing stage. This significantly reduces computational complexity while preserving the structural content of the image. No filtering operations are applied during this stage, ensuring the data remains in its original, unblurred form to preserve sharp edges for accurate detection.

After preprocessing, the grayscale image is passed to the 2D convolution block. Here, the image undergoes convolution with predefined kernels designed for edge detection, such as the Sobel, Prewitt, or Roberts operators. These convolution kernels are typically 3×3 matrices that detect intensity gradients in horizontal and vertical directions. The convolution operation calculates the gradient magnitude by applying these kernels over the entire image in a sliding window manner. On FPGA, this is efficiently achieved using line buffers and Multiply-Accumulate (MAC) units, with pipelining to enable simultaneous computation across image pixels. The ability of FPGAs to handle multiple convolution operations in parallel allows for rapid and efficient processing suitable for real-time applications.

The output of the 2D convolution operation is a gradient image, which is then processed in the edge detection stage. Here, a thresholding technique is applied to identify significant gradients that correspond to edges. Pixels with gradient magnitudes above a certain threshold are classified as edge pixels, while those below are discarded or set to zero. This produces a binary edge map, where edges are clearly defined. In some implementations, additional steps such as gradient direction analysis or edge thinning can be added, but in basic designs, simple thresholding is used to keep the hardware resource usage minimal and the speed optimal.

Finally, the result is presented as an edges highlighted image, where the detected edges are emphasized, typically as white pixels on a black background. This final output can be displayed on a monitor, sent to memory, or used in downstream systems like object recognition or navigation modules. The FPGA's architecture allows seamless integration of the entire pipeline—from image acquisition and preprocessing to convolution and edge detection—on a single chip. By exploiting parallel data paths, hardware accelerators, and on-chip memory blocks, the system is capable of performing high-speed edge detection without introducing significant latency, making it ideal for time-sensitive applications.

3.2 Morphological Chart

The morphological chart for the Edge Detection System on the PYNQ-Z2 board provides a structured overview of the key functional blocks and their respective design options. It serves as a tool to systematically explore different implementation choices and their trade-offs, helping guide the design process.

- **Image Input:** The system can accept various image sources, such as stored test images for offline processing, real-time webcam feeds for live experimentation, or

Function	Option 1	Option 2	Option 3
Image Input	Stored test image	Real-time webcam	HDMI input (live stream)
Pre-processing	RGB to Grayscale (software)	Grayscale in hardware (HLS)	Native grayscale camera
Edge Detection Algorithm	Sobel filter	Prewitt filter	Canny filter
Implementation Platform	Vitis HLS hardware	Python on ARM Cortex-A9	OpenCV with PYNQ overlay
Data Transfer	AXI-Stream with DMA	AXI Memory-Mapped	Direct CPU control

Table 3.1: Comparison of Implementation Options

HDMI input for capturing high-definition video streams directly.

- **Pre-processing:** Pre-processing converts raw image data into a suitable format for edge detection. This can be done in software by converting RGB images to grayscale, or in hardware using high-level synthesis (HLS) to accelerate grayscale conversion. Alternatively, a native grayscale camera may be used to simplify processing.

- **Edge Detection Algorithm:** Different algorithms can be employed depending on the complexity and accuracy requirements. The Sobel filter is a common choice for its simplicity and hardware efficiency, while Prewitt offers a similar approach with different kernel weights. The Canny filter provides more precise edge detection at the cost of higher computational complexity.

- **Implementation Platform:** The design can be implemented as hardware accelerators using Vitis HLS for FPGA logic, in software running on the ARM Cortex-A9 processor, or via Python and OpenCV leveraging PYNQ overlays for rapid prototyping.

- **Data Transfer:** Efficient data movement between the processor and FPGA is crucial. Options include AXI-Stream interfaces with DMA for high-throughput streaming, AXI Memory-Mapped transfers for controlled access, or direct CPU control for simpler but slower interactions.

This morphological chart encapsulates the trade-offs and flexibility in system design, enabling informed decisions to optimize performance, resource utilization, and development complexity for the edge detection application on the PYNQ-Z2 platform.

3.3 Design Alternatives

In developing the edge detection system on the PYNQ-Z2 platform, multiple design alternatives were considered at various stages of the project to balance performance, complexity, and resource utilization.

Image Input Alternatives: While a stored test image simplifies initial development and debugging, image acquisition options such as a USB webcam or HDMI video input offer greater application flexibility. The USB webcam provides easy connectivity and wide support, but may introduce latency and driver dependencies. HDMI input allows higher resolution and direct video streaming but requires more complex hardware interfacing and increased FPGA resource usage.

Pre-processing Approaches: The RGB to grayscale conversion can be performed either in software on the ARM processor or in hardware using FPGA logic. Software conversion offers easier implementation and rapid prototyping but may become a bottleneck for high-resolution processing. Hardware-based conversion accelerates this step and reduces CPU load but increases design complexity and FPGA resource consumption.

Edge Detection Algorithms: Several edge detection methods were evaluated including Sobel, Prewitt, and Canny filters. The Sobel filter was ultimately chosen for its relative simplicity and suitability for hardware implementation. Prewitt offers similar performance with slightly less sensitivity to noise, whereas Canny provides superior edge detection quality at the cost of increased computational complexity, making it less practical for resource-constrained FPGA deployment.

Implementation Platforms: The system could be implemented fully in software on the ARM Cortex-A9 or partially in hardware using Vitis HLS for the computationally intensive parts. Software-only implementation is simpler and more flexible but suffers from lower performance. Hardware acceleration using FPGA fabric significantly improves throughput and latency but requires expertise in hardware design and synthesis.

Data Transfer Mechanisms: Data transfer between the processor and programmable logic can be handled via AXI-Stream with DMA, AXI Memory-Mapped interfaces, or direct CPU control. AXI-Stream with DMA is preferred for high-speed streaming data as it minimizes CPU intervention, whereas memory-mapped interfaces are simpler but less efficient for continuous data flows. Direct CPU control provides flexibility but introduces latency and increases processor overhead.

Each alternative carries trade-offs in terms of development time, performance, hardware resource utilization, and system complexity. The final design choices reflect a balanced approach that meets the project goals of real-time processing, hardware-software co-design, and efficient use of the PYNQ-Z2 platform's capabilities.

3.4 Design Choices and Justifications

For the Edge Detection System implemented on the PYNQ-Z2 board, the following design choices were made based on performance, resource constraints, and development efficiency:

- **Image Input:** A stored test image was chosen for initial development and testing to simplify debugging and verification. Real-time webcam input is planned for future iterations to enable live processing, but static images allow controlled experimentation.

- **Pre-processing:** The RGB to grayscale conversion was implemented in software initially to leverage the flexibility of Python and the ARM Cortex-A9 processor. However, to accelerate the pipeline, grayscale conversion was also moved to hardware using Vitis HLS, enabling faster processing and offloading the CPU.

- **Edge Detection Algorithm:** The Sobel filter was selected for its simplicity and suitability for hardware implementation. It strikes a balance between computational complexity and edge detection quality, making it ideal for FPGA acceleration on the PYNQ-Z2.

- **Implementation Platform:** The core edge detection logic was synthesized into hardware using Vitis HLS, allowing parallel processing on the FPGA fabric. Control and data transfer are managed by software running on the ARM Cortex-A9, exploiting the heterogeneous architecture of the Zynq SoC for efficient hardware-software co-design.
- **Data Transfer:** AXI-Stream interfaces with DMA were chosen to facilitate high-throughput, low-latency data transfer between the processor and programmable logic. This ensures efficient streaming of image data for real-time processing without burdening the CPU with data movement tasks.

These choices optimize system performance while leveraging the strengths of the PYNQ-Z2 board's hardware and software ecosystem, enabling a flexible, efficient, and scalable edge detection solution.

Chapter 4

Implementation details

The edge detection algorithm was developed using a high-level programming language and converted into hardware logic through synthesis tools. The hardware module was integrated with a processing system to enable seamless data exchange. The system runs an embedded operating environment that manages hardware control and data flow. The detailed implementation details is as follows:

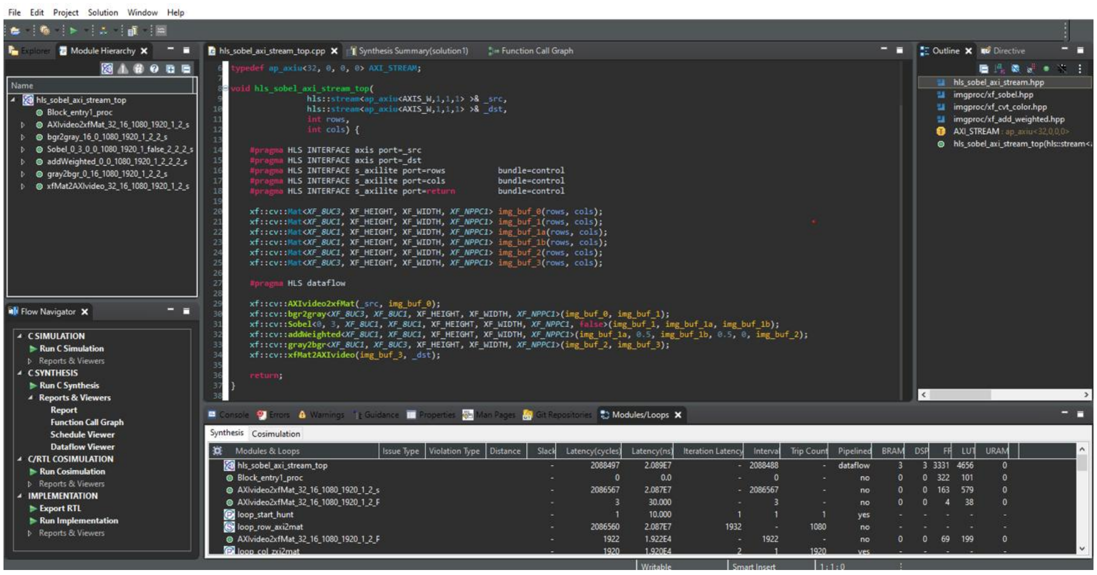
4.1 Edge Detection Algorithm Using 2D Convolution Implemented on PYNQ-Z2 FPGA Platform

Step	**Description**
1.	Input an RGB image.
2.	Convert the RGB image to grayscale using the weighted sum.
3.	Send the grayscale image data to the FPGA via AXI-Stream using DMA.
4.	In FPGA (hardware logic), apply the Sobel operator using two 3x3 convolution kernels to calculate horizontal (G_x) and vertical (G_y) gradients.
5.	Compute the gradient magnitude at each pixel.
6.	Compare the gradient magnitude with a fixed threshold value: If value > threshold, mark as edge. Otherwise, the pixel is considered part of the background.
7.	Display the final edge-detected image.

Table 4.1: Steps for FPGA-based Edge Detection using Sobel Operator

Hardware Logic Design and High-Level Synthesis

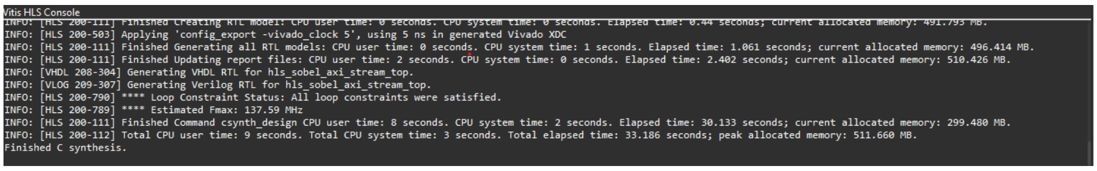
The core logic of the Sobel edge detection filter is initially developed and verified using **Vitis High-Level Synthesis (HLS)** as shown in figure 4.1 and 4.2, a tool that allows algorithms described in high-level C or C++ code to be synthesized into Register-Transfer Level (RTL) hardware implementations. During this phase, the Sobel filter algorithm is implemented algorithmically, including the convolution with Sobel kernels, gradient magnitude calculations, and thresholding steps. Vitis HLS enables functional simulation and performance verification of the design using test-benches before generating synthesizable Verilog or VHDL code.



```
typedef ap_axiu<32, 0, 0, 0> AXI_STREAM;

void hls_sobel_axi_stream_top(
            hls::stream<ap_axiu<AXIS_W,1,1,1> >& _src,
            hls::stream<ap_axiu<AXIS_W,1,1,1> >& _dst,
            int rows,
            int cols) {

    #pragma HLS INTERFACE axis port=_src
    #pragma HLS INTERFACE axis port=_dst
    #pragma HLS INTERFACE s_axilite port=rows          bundle=control
    #pragma HLS INTERFACE s_axilite port=cols          bundle=control
    #pragma HLS INTERFACE s_axilite port=return        bundle=control

    xf::cv::Mat<XF_8UC3, XF_HEIGHT, XF_WIDTH, XF_NPPC1> img_buf_0(rows, cols);
    xf::cv::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> img_buf_1(rows, cols);
    xf::cv::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> img_buf_1a(rows, cols);
    xf::cv::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> img_buf_1b(rows, cols);
    xf::cv::Mat<XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1> img_buf_2(rows, cols);
    xf::cv::Mat<XF_8UC3, XF_HEIGHT, XF_WIDTH, XF_NPPC1> img_buf_3(rows, cols);

    #pragma HLS dataflow

    xf::cv::AXIvideo2xfMat(_src, img_buf_0);
    xf::cv::bgr2gray<XF_8UC3, XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1>(img_buf_0, img_buf_1);
    xf::cv::Sobel<0, 3, XF_8UC1, XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1, false>(img_buf_1, img_buf_1a, img_buf_1b);
    xf::cv::addWeighted<XF_8UC1, XF_8UC1, XF_HEIGHT, XF_WIDTH, XF_NPPC1>(img_buf_1a, 0.5, img_buf_1b, 0.5, 0, img_buf_2);
    xf::cv::gray2bgr<XF_8UC1, XF_8UC3, XF_HEIGHT, XF_WIDTH, XF_NPPC1>(img_buf_2, img_buf_3);
    xf::cv::xfMat2AXIvideo(img_buf_3, _dst);

    return;
}
```

Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
hls_sobel_axi_stream_top				-	2088497	2.089E7	-	2088488	-	dataflow	3	3	3331	4656	0
Block_entry1_proc				-	0	0.0	-	0	-	no	0	0	322	101	0
AXIvideo2xfMat_32_16_1080_1920_1_2_s				-	2086567	2.087E7	-	2086567	-	no	0	0	163	579	0
AXIvideo2xfMat_32_16_1080_1920_1_2_F				-	3	30.000	-	3	-	no	0	0	4	38	0
loop_start_hunt				-	1	10.000	1	1	1	yes	-	-	-	-	-
loop_row_axi2mat				-	2086560	2.087E7	1932	-	1080	no	-	-	-	-	-
AXIvideo2xfMat_32_16_1080_1920_1_2_F				-	1922	1.922E4	-	1922	-	no	0	0	69	199	0
loop_col_zxi2mat				-	1920	1.920E4	2	1	1920	yes	-	-	-	-	-

Figure 4.1: Logic of Sobel edge detection filter using 2D Convolution implemented in Vitis HLS



```
INFO: [HLS 200-111] Finished Creating RTL model: CPU user time: 0 seconds. CPU system time: 0 seconds. Elapsed time: 0.44 seconds; current allocated memory: 491.793 MB.
INFO: [HLS 200-503] Applying 'config_export -vivado_clock 5', using 5 ns in generated Vivado XDC
INFO: [HLS 200-111] Finished Generating all RTL models: CPU user time: 0 seconds. CPU system time: 1 seconds. Elapsed time: 1.061 seconds; current allocated memory: 496.414 MB.
INFO: [HLS 200-111] Finished Updating report files: CPU user time: 2 seconds. CPU system time: 0 seconds. Elapsed time: 2.402 seconds; current allocated memory: 510.426 MB.
INFO: [VHDL 208-304] Generating VHDL RTL for hls_sobel_axi_stream_top.
INFO: [VLOG 209-307] Generating Verilog RTL for hls_sobel_axi_stream_top.
INFO: [HLS 200-790] **** Loop Constraint Status: All loop constraints were satisfied.
INFO: [HLS 200-789] **** Estimated Fmax: 137.59 MHz
INFO: [HLS 200-111] Finished Command csynth_design CPU user time: 8 seconds. CPU system time: 2 seconds. Elapsed time: 30.133 seconds; current allocated memory: 299.480 MB.
INFO: [HLS 200-112] Total CPU user time: 9 seconds. Total CPU system time: 3 seconds. Total elapsed time: 33.186 seconds; peak allocated memory: 511.660 MB.
Finished C synthesis.
```

Figure 4.2: Performance parameters of the developed sobel logic

Once the algorithm's functionality and timing are validated, the Sobel filter logic is exported from Vitis HLS as a reusable **Intellectual Property (IP) core**. This IP core includes detailed metadata defining interfaces such as AXI4-Stream or AXI4-Lite, clocking, and control signals. Exporting as an IP abstracts the complex hardware details, allowing it to be seamlessly integrated into larger hardware designs without manual handling of low-level signal interfacing.

Hardware Integration Using Vivado Design Suite

The exported Sobel IP core is imported into the **Vivado Design Suite**, where the complete hardware system is constructed within a block design environment as shown in figure 4.3. Here, the Sobel IP is integrated with other essential components including the **ZYNQ7 Processing System** (representing the ARM Cortex-A9), **AXI interconnects**, and a **Direct Memory Access (DMA) controller**. These IP blocks facilitate efficient communication between the processing system (PS) and the programmable logic (PL).

The AXI DMA controller plays a critical role in transferring large image data between the processor and FPGA fabric efficiently, offloading the CPU from intensive data movement tasks. Within Vivado, proper address mapping, clock synchronization, and port

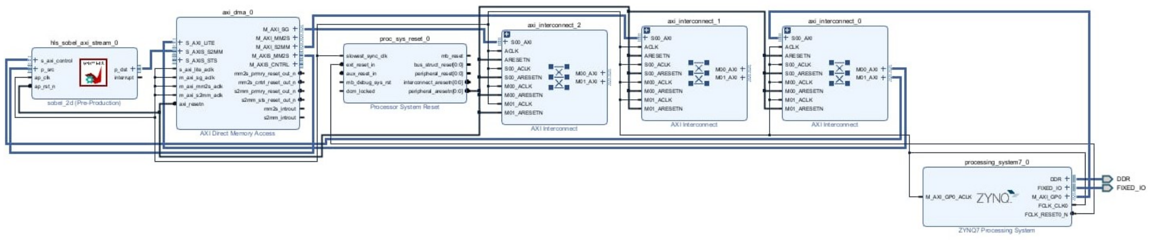


Figure 4.3: Block design in Vivado Design Suite

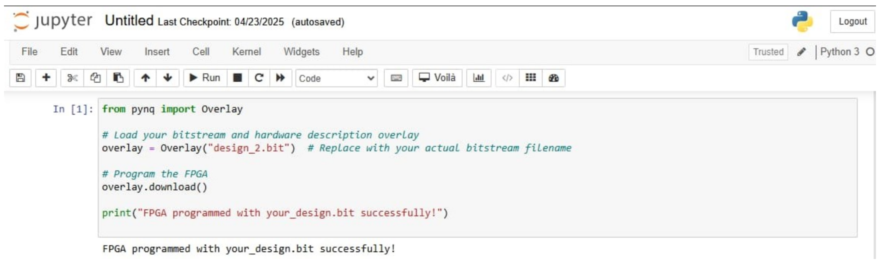
configuration are managed to ensure smooth data flow and enable hardware-software co-design. After integration, the entire hardware design undergoes simulation and verification to confirm functionality.

Following validation, Vivado generates an **HDL wrapper** to encapsulate the block design as a top-level module, then produces the **bitstream (.bit)** file to configure the FPGA, the **hardware handoff (.hwh)** file describing the hardware interfaces, and a **TCL script (.tcl)** to facilitate design reuse or automation. These files are essential for programming and controlling the hardware system on the target platform.

Deployment and Runtime Environment on PYNQ-Z2 Board

The deployment process moves to the **PYNQ-Z2 board**, which combines a Xilinx Zynq-7000 SoC with an embedded Linux environment and supports Python scripting. The board is connected via USB or Ethernet, with Ethernet preferred for faster data transfer and access to the Jupyter Notebook server running on the board.

After powering the board, an IP address is assigned either dynamically via DHCP or statically by the user. Using PuTTY or any serial terminal emulator, the developer accesses the board's Linux shell to verify network connectivity. The Jupyter Notebook interface is then accessed by entering the board's IP address in a web browser. Within Jupyter,



```
In [1]: from pynq import Overlay

        # Load your bitstream and hardware description overlay
        overlay = Overlay("design_2.bit")  # Replace with your actual bitstream filename

        # Program the FPGA
        overlay.download()

        print("FPGA programmed with your_design.bit successfully!")

        FPGA programmed with your_design.bit successfully!
```

Figure 4.4: Deployment of the design onto the hardware using Jupyter Notebook

the bitstream, hardware handoff, and TCL script are uploaded. Using the PYNQ Python API's **Overlay** class, the Sobel IP core is dynamically loaded into the FPGA fabric as shown in the figure 4.4. The ARM Cortex-A9 processor handles control, input/output management, and initiates the hardware logic execution through memory-mapped interfaces, effectively bridging software and hardware tasks.

Performance Validation and Benchmarking

With the system deployed, performance validation compares the Sobel edge detection algorithm running in two different environments:

- **Software-only execution** on the ARM Cortex-A9 processor.
- **Hardware-accelerated execution** on the FPGA programmable logic using the custom Sobel IP.

The software version implements convolution operations sequentially using Python or C/C++, while the hardware version leverages the parallelism and dedicated logic resources of the FPGA. Data transfers are handled by DMA and AXI interconnects to maximize throughput.

Execution times are measured and compared using profiling tools and built-in timers. The results consistently show that hardware acceleration provides significantly lower latency and higher throughput, especially as image resolution increases or real-time processing demands grow. This clearly demonstrates the advantages of offloading computationally intensive image processing tasks to programmable logic, enabling efficient, high-speed edge detection.

Conclusion

This project exemplifies a complete hardware-software co-design workflow, starting from high-level algorithm development in Vitis HLS, integration and verification in Vivado, and culminating in deployment and runtime testing on the PYNQ-Z2 platform. The work highlights the Zynq architecture's strengths in enabling real-time embedded vision systems and confirms the effectiveness of FPGA-based acceleration for image processing applications such as smart cameras, autonomous vehicles, and embedded AI systems.

4.2 Flowchart

The process of edge detection on an FPGA as shown in figure 4.5 begins with the acquisition of a digital RGB input image, typically from a camera module or image sensor. Since edge detection focuses on intensity transitions rather than color, the first step is to convert the RGB image to grayscale. This conversion reduces the data to a single intensity channel, which simplifies the processing pipeline and reduces hardware resource usage without significantly affecting the detection of edge features. Once the grayscale image is obtained, it is subjected to 2D convolution using the Sobel operator. The Sobel operator applies two 3×3 convolution kernels—Gx and Gy—to estimate the gradient in the horizontal and vertical directions, respectively. This operation is performed pixel-by-pixel, where a small kernel window slides over the image and computes a weighted sum of neighboring pixel intensities. The convolution with Gx detects vertical edges, while Gy identifies horizontal edges. These directional gradients are essential for evaluating the strength and orientation of edges.

Following convolution, the system computes the gradient magnitude at each pixel using the results of Gx and Gy. The gradient magnitude is calculated as the square root of the sum of the squares of the horizontal and vertical gradients, or it can be approximated for faster computation using the sum of absolute values This magnitude reflects how sharp

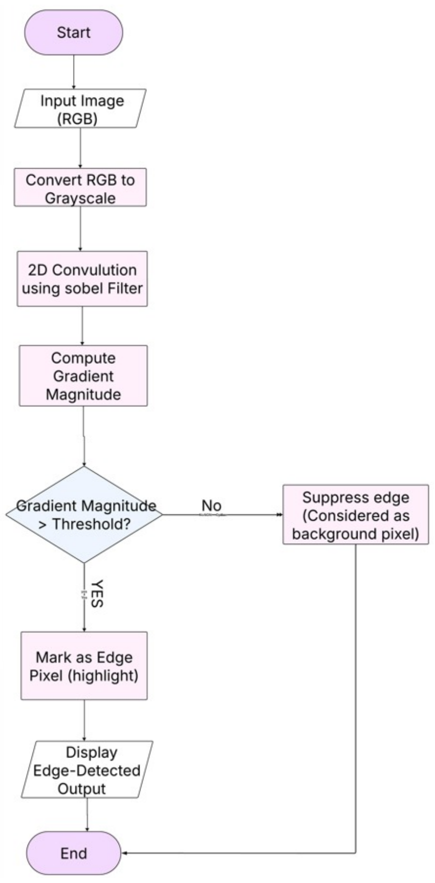

Start
Input Image (RGB)
Convert RGB to Grayscale
2D Convulution using sobel Filter
Compute Gradient Magnitude
Gradient Magnitude > Threshold?
No
Suppress edge (Considered as background pixel)
YES
Mark as Edge Pixel (highlight)
Display Edge-Detected Output
End

Figure 4.5: Flowchart representing real-time edge detection implemented on software and hardware

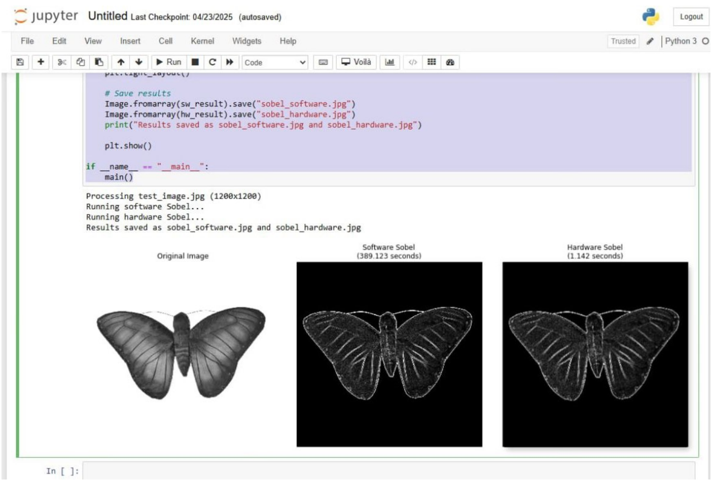
the intensity transition is at a given pixel and is used to determine the presence of an edge.

To distinguish actual edges from background noise, the computed gradient magnitude is compared to a predefined threshold. If the gradient value exceeds the threshold, the pixel is marked as an edge pixel and highlighted in the output image. Otherwise, the pixel is suppressed and treated as a background pixel, ensuring that only significant edges are retained. This thresholding step is crucial for reducing false positives and enhancing the clarity of the edge-detected output.

Finally, the system displays or outputs the edge-detected image, in which only the strong and relevant edges are emphasized. This output can be used for further analysis in applications such as object detection, navigation, or scene understanding. The entire process is highly optimized for real-time FPGA implementation, leveraging the parallelism and reconfigurability of hardware to perform high-speed, low-latency image processing. Tasks such as convolution, magnitude calculation, and thresholding are pipelined and executed concurrently, making the system ideal for applications like Advanced Driver Assistance Systems (ADAS), where rapid and accurate edge detection is critical for safety and performance.

4.3 Results

(a) Input image

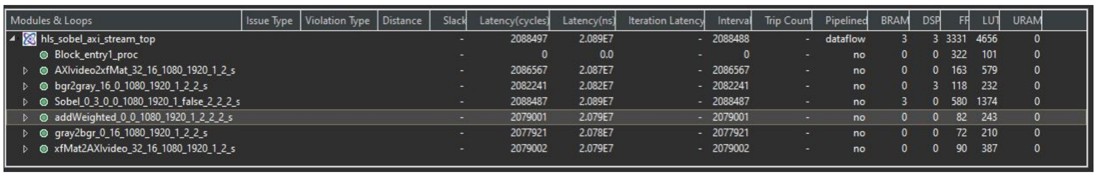


```
    # Save results
    Image.fromarray(sw_result).save("sobel_software.jpg")
    Image.fromarray(hw_result).save("sobel_hardware.jpg")
    print("Results saved as sobel_software.jpg and sobel_hardware.jpg")

    plt.show()

if __name__ == "__main__":
    main()
```

```
Processing test_image.jpg (1200x1200)
Running software Sobel...
Running hardware Sobel...
Results saved as sobel_software.jpg and sobel_hardware.jpg
```

(b) Real-time edge detection

Figure 4.6: Comparison of input and edge detection results



Modules & Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
hls_sobel_axi_stream_top				-	2088497	2.089E7	-	2088488	-	dataflow	3	3	3331	4656	0
Block_entry1_proc				-	0	0.0	-	0	-	no	0	0	322	101	0
AXIvideo2xfMat_32_16_1080_1920_1_2_s				-	2086567	2.087E7	-	2086567	-	no	0	0	163	579	0
bgr2gray_16_0_1080_1920_1_2_2_s				-	2082241	2.082E7	-	2082241	-	no	0	3	118	232	0
Sobel_0_3_0_0_1080_1920_1_false_2_2_2_s				-	2088487	2.089E7	-	2088487	-	no	3	0	580	1374	0
addWeighted_0_0_1080_1920_1_2_2_2_s				-	2079001	2.079E7	-	2079001	-	no	0	0	82	243	0
gray2bgr_0_16_1080_1920_1_2_2_s				-	2077921	2.078E7	-	2077921	-	no	0	0	72	210	0
xfMat2AXIvideo_32_16_1080_1920_1_2_s				-	2079002	2.079E7	-	2079002	-	no	0	0	90	387	0

Figure 4.7: High-Level Synthesis (HLS) report showing hardware resource utilization for each module in the Sobel Edge Detection pipeline on PYNQ-Z2.

Figure 4.6a [18] illustrates the original input image that is fed into the edge detection system. This image is initially in RGB color format and serves as the primary visual data source for processing. Prior to edge detection, the RGB image undergoes a color space conversion to grayscale to simplify computations and enhance edge visibility, as most edge detection algorithms operate on single-channel intensity images. Figure 4.7 displays the result of the edge detection process applied to the grayscale version of the input image. This output demonstrates the system's ability to extract prominent edges in real time, highlighting the boundaries and features within the original image. The figure showcases the effectiveness of both software and hardware implementations of the edge detection pipeline, confirming the successful transformation from raw input to processed edge map.

The High-Level Synthesis (HLS) report provides a detailed analysis of latency and hardware resource utilization for each module in the Sobel edge detection pipeline implemented on the PYNQ-Z2 FPGA. The modules include color format conversions (bgr2gray and gray2bgr), the Sobel filter, and AXI stream interfaces. Among these, the Sobel module—responsible for 2D convolution and gradient computation—shows the highest resource usage, consuming 1374 LUTs, 580 flip-flops, and 3 BRAM blocks to support sliding window operations over high-resolution images. In contrast, modules like bgr2gray make use of 3 DSP blocks for grayscale conversion due to arithmetic operations involving

Table 4.2: Summary of Resource Usage and Latency

Resource	**Quantity / Value**
DSP Blocks	3
BRAM Blocks	3
LUTs	1374
Flip-Flops (FFs)	580
Latency (per module)	2 million clock cycles

floating-point coefficients. Notably, none of the modules are pipelined, which contributes to the high latency values observed (over 2 million clock cycles per module). The AXI interface modules show negligible hardware usage and zero latency, as they primarily handle data format conversion. Overall, the report highlights the Sobel operation as the most resource-intensive stage, suggesting that performance can be significantly improved through pipelining and optimized resource sharing.

Chapter 5

Conclusions and Future Scope

This project successfully implemented real-time edge detection on the PYNQ-Z2 board using the Sobel filter and FPGA-based acceleration, achieving improved speed and performance. The hardware-software co-design approach ensured both flexibility and computational efficiency. The system can be enhanced with advanced algorithms, higher-resolution processing, and integration with machine learning. It has significant applications in domains like autonomous driving, healthcare, and surveillance. Detailed insights into the conclusion, future scope, and societal applications are provided below.

5.1 Conclusion

This project successfully demonstrated the design and implementation of a real-time edge detection system on the PYNQ-Z2 platform. Leveraging the hybrid architecture of the Zynq-7000 SoC, the system combined the flexibility of software running on the ARM Cortex-A9 processor with the computational power of FPGA-based hardware acceleration. The Sobel filter, implemented as a reusable IP core via Vitis HLS, provided an efficient and reliable method for detecting edges in grayscale images. Integration of the hardware IP with other necessary components using Vivado, and deployment on the PYNQ-Z2 board using Python and Jupyter Notebooks, showcased an effective hardware-software co-design methodology.

Performance comparisons highlighted the significant speed-up achieved through FPGA acceleration compared to a software-only implementation on the ARM processor. This improvement validates the advantage of using programmable logic for compute-intensive image processing tasks. Additionally, the project explored multiple design alternatives and made informed choices that balanced resource utilization, ease of development, and real-time performance. Overall, the project establishes a foundation for further exploration of real-time image processing applications on embedded FPGA platforms.

5.2 Future Scope

The real-time edge detection system developed here can be extended and enhanced in various ways. Future work could focus on implementing more advanced edge detection algorithms such as the Canny filter, which, despite higher complexity, offers improved accuracy and noise resilience. Optimization of hardware resources could allow processing of higher-resolution images and video streams at faster frame rates. Moreover, the integration of additional image processing tasks like noise reduction, segmentation, or object recognition would expand the system's applicability.

From a system perspective, incorporating machine learning models onto the programmable logic for tasks such as feature extraction or classification could lead to a versatile embedded vision system. The current setup could also be adapted to support multiple camera inputs or interface with other sensors to enable more complex smart applications. Finally, improvements in data transfer mechanisms and driver development could enhance throughput and user experience.

5.3 Application in the Societal Context

Edge detection and related image processing techniques play a crucial role in numerous societal applications. In the domain of autonomous vehicles and driver assistance systems, real-time edge detection facilitates accurate lane detection and obstacle recognition, contributing to safer and smarter transportation.

In surveillance and security, real-time edge detection supports the identification of suspicious activities and the enhancement of video analytics. Additionally, in environmental monitoring and agriculture, such techniques help in analyzing satellite and drone imagery for vegetation mapping, land use classification, and disaster management.

By implementing these algorithms on embedded platforms like the PYNQ-Z2, systems can be deployed in resource-constrained or remote environments, making advanced image processing accessible and affordable. This democratization of technology holds the potential to impact public health, safety, and environmental sustainability positively. Therefore, the project's outcomes can serve as building blocks for future innovations that address real-world societal challenges through intelligent, efficient, and embedded vision systems.

Bibliography

[1] Tulip Bera and Shuvabrata Dey. Fpga implementation of systolic array for 2d convolution using residue number system. In *2023 International Conference on Electrical, Electronics, Communication and Computers (ELEXCOM)*, pages 1–6. IEEE, 2023.

[2] All About Circuits. Two-dimensional convolution in image processing, 2019. Accessed: 2025-06-02.

[3] Dfrobot pynq-z2 development board. https://www.mouser.in/new/dfrobot/dfrobot-pynqz2-dev-board/, 2025. Accessed: 2025-06-02.

[4] Bestami Günay, Sefa Burak Okcu, and Hasan Şakir Bilge. Lpyolo: low precision yolo for face detection on fpga. *arXiv preprint arXiv:2207.10482*, 2022.

[5] Laigong Guo and Sitong Wu. Fpga implementation of a real-time edge detection system based on an improved canny algorithm. *Applied sciences*, 13(2):870, 2023.

[6] Agrim Gupta and Viktor K Prasanna. Energy efficient image convolution on fpga. *Viterbi India 2013 Program*, (1), 2013.

[7] Md Jahiruzzaman, Shumit Saha, and Md Abul Khayum Hawlader. Dynamically reconfigurable parallel architecture implementation of 2d convolution for image processing over fpga. In *2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)*, pages 1–6. IEEE, 2015.

[8] Gian Domenico Licciardo, Carmine Cappetta, and Luigi Di Benedetto. Fpga optimization of convolution-based 2d filtering processor for image processing. In *2016 8th Computer Science and Electronic Engineering (CEEC)*, pages 180–185. IEEE, 2016.

[9] Ákos Mándi, Jeney Máté, Dominik Rózsa, and Stefan Oniga. Hardware accelerated image processing on fpga based pynq-z2 board. *Carpathian journal of electronic and computer engineering*, 14(1):20–23, 2021.

[10] Rajesh Mehra and Rupinder Verma. Area efficient fpga implementation of sobel edge detector for image processing applications. *International Journal of Computer Applications*, 56(16), 2012.

[11] Sourabh Mehta. Different edge detection techniques with implementation in opencv. *Analytics India Magazine*, 2022. Accessed: 2025-06-02.

[12] OpenCV Development Team. Sobel derivatives — opencv documentation. https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html, 2023. Accessed: 2025-06-02.

[13] Thaufig Peng-o and Panyayot Chaikan. High performance and energy efficient sobel edge detection. *Microprocessors and Microsystems*, 87:104368, 2021.

[14] Trung Dinh Pham, Bao Gia Bach, Lam Trinh Luu, Minh Dinh Nguyen, Hai Duc Pham, Khoa Bui Anh, Xuan Quang Nguyen, and Cuong Pham Quoc. An fpga-based solution for convolution operation acceleration. *arXiv preprint arXiv:2206.04520*, 2022.

[15] Real Digital. Pynq-z2 development board, 2025. Accessed: 2025-06-02.

[16] Jonathan Sanderson and Syed Rafay Hasan. System integration of xilinx dpu and hdmi for real-time inference in pynq environment with image enhancement. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2024.

[17] Manupoti Sreenivasulu and T Meenpal. Efficient hardware implementation of 2d convolution on fpga for image processing application. In *2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–5. IEEE, 2019.

[18] Wild Republic. Finger puppets collection, 2025. Accessed: 2025-06-04.

[19] Jinmei Zhang, Zhangyao Zi, Tao Jiang, Chao Zhang, and Yonghang Tai. Implementation and optimization of fpga-based edge detection algorithm. In *Advancements in Mechatronics and Intelligent Robotics: Proceedings of ICMIR 2020*, pages 611–617. Springer, 2021.