EDGE DETECTION USING PYNQ-Z2 BOARD

EEC51806 – PROJECT PHASE- I

Submitted by

P. VAMSI KRISHNA	**22119006**
M. DURGA SIVA KRISHNA	**22119019**
U. BHAVANI SHANKAR	**22119018**

Under the guidance of

Dr. M. Rajmohan

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

HINDUSTAN INSTITUTE OF TECHNOLOGY AND SCIENCE

CHENNAI – 603 103

NOVEMBER 2025

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

BONAFIDE CERTIFICATE

Certified that this Phase I project report **"EDGE DETECTION USING PYNQ-Z2 BOARD"** is the bonafide work of **P. VAMSI KRISHNA (22119006), M. DURGA SIVA KRISHNA (22119019) & U. BHAVANI SHANKAR (22119018)** carried out the project workunder my supervision during the academic year 2025-2026.

SIGNATURE
HEAD OF THE DEPARTMENT
Dr. Al. Vallikannu
Professor and Head of the Department
Department of Electronics and Communication Engineering
Hindustan Institute of Technology and Science
Padur, Chennai-603103

SIGNATURE
SUPERVISOR
Dr. M. Rajmohan
Associate Professor
Department of Electronics and Communication Engineering
Hindustan Institute of Technology and Science
Padur, Chennai,603103

INTERNAL EXAMINER
Name: _____
Designation: _____
Project Viva-Voce conducted on _____

EXTERNAL EXAMINER
Name: _____
Designation: _____
Institution Name: _____

ACKNOWLEDGEMENT

It is our extreme pleasure to thank our Chancellor **Dr. Anand Jacob Verghese**, Pro Chancellor **Dr. Ashok George Verghese**, Interim Vice Chancellor **Dr. S.Ganesan** , Hindustan Institute ofTechnology and Science for providing a conducive environment that helped us to pursue our project in a diligent and efficient manner. We wish to express our sincere gratitude to Dean (Academics) **Dr. Angelina Geetha**, Hindustan Institute of Technology and Science for her valuable directions, suggestions, and support.

We are thankful to **Dr. AL Vallikannu**, Head of the Department, Electronics and Communication Engineering for having evinced keen interest in our project andfor her continued support.

We are indebted to our Project Coordinator **Dr**. **E. Terence Ebinesan**, Associate Professor of Electronics and Communication Engineering and Project guide **Dr. M. Rajmohan** for his guidance and technical support in the accomplishment of our project.

Our sincere thanks to all the teaching and non-teaching staff and family members who have been constantly supporting us throughout the accomplishment of this project.

ABSTRACT

This project presents the design and hardware implementation of a real-time edge detection system based on the Sobel filter on the PYNQ-Z2 FPGA platform. The core objective is to accelerate the computationally intensive process of edge detection by developing custom Sobel X and Sobel Y filter IP cores using VHDL, and integrating them with AXI DMA and SmartConnect infrastructure in Vivado. The implemented hardware pipeline processes grayscale images from a webcam, performs parallel convolution in both horizontal and vertical directions, and transfers filtered results between the programmable logic and processing system. Simulation and synthesis demonstrate not only the correctness of the edge detection algorithm but also a marked speedup over conventional software execution. The validated design lays the groundwork for integration with Python and Jupyter-based real-time demonstrations, providing a robust foundation for high-speed image processing in embedded systems and highlighting the advantages of FPGA acceleration for vision applications.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

Image processing is a rapidly growing field with applications ranging from medical diagnostics and forensics to autonomous vehicles, industrial automation, surveillance, and robotics. One of the key operations in image analysis is edge detection, which identifies significant transitions in pixel values corresponding to object boundaries and salient features within an image. Traditional image processing methods implemented on general-purpose CPUs or GPUs often face challenges in meeting the strict real-time and low-latency requirements of these applications, especially when processing high-resolution or streaming video data.

Field Programmable Gate Arrays (FPGAs) provide a powerful platform for accelerating computation-intensive image processing tasks such as convolution, filtering, and edge detection. FPGAs offer massive parallelism, customizable architecture, and efficient on-chip memory utilization, which enable low-latency and energy-efficient solutions for embedded real-time systems. Implementing edge detection algorithms like the Sobel operator on FPGA allows direct hardware-level pixel-level processing, reducing processing delays and enhancing reliability for mission-critical tasks.

Given the increasing demand for high-performance yet flexible image processing in fields requiring immediate responses, there is a clear need to develop FPGA-based hardware accelerators that can meet both speed and adaptability requirements. The integration of such accelerators, exemplified by edge detection IP cores on modern FPGA platforms like PYNQ-Z2, directly addresses these challenges, providing a foundation for scalable, real-world embedded vision solutions.

1.2 Significance

a) Enables real-time edge detection with high speed and low latency through FPGA parallelism.

b) Provides greater power efficiency compared to software or GPU implementations, suitable for embedded systems.

c) Improves accuracy and speed of image processing for applications like robotics, surveillance, and medical imaging.

d) Allows customized hardware acceleration tailored to Sobel filtering, reducing computational overhead.

e) Shows a scalable approach for implementing complex image processing tasks on FPGA platforms.

1.3 Objective of the Project

a) This project implements real-time image edge detection using Sobel filters accelerated in hardware on a Pynq-Z2 FPGA board.
b) Design and develop VHDL-based hardware IPs (Sobel X and Sobel Y) with AXI-Stream compatibility.
c) Integrate the IPs into Vivado block design with DMA, AXI SmartConnect, and PS–PL interfaces.

1.4 Methodology

a) Designing Custom Sobel IP Cores: Implementing the Sobel X and Y convolution kernels as efficient hardware IPs required accurate VHDL coding, careful pipelining, and correct AXI-Stream interface integration. Bugs in the initial filter logic occasionally led to incorrect output patterns, requiring iterative debugging and verification.

b) Configuring the Vivado Block Design: Integrating multiple custom and standard IP blocks—such as DMA engines, AXI interconnects, and the Zynq Processing System—demanded detailed attention to interface signals, clock domain crossings, and proper addressing for seamless data movement between modules.

c) Simulation and Functional Verification: Creating reliable testbenches to simulate the Sobel filter and peripheral interconnects was challenging, especially in ensuring test vectors replicated realistic pixel streams and timing conditions. Early testbenches sometimes failed to catch edge cases, leading to unexpected results in simulation.

d) Resource Utilization and Timing: After synthesis, balancing logic resource usage and ensuring the design met timing constraints was crucial. Over-constrained or under-optimized designs could fail implementation or reduce achievable clock frequency, impacting performance.

1.5 Organization of the Report

The report is organized into several chapters to provide a structured and comprehensive overview of the project. It begins with an introduction detailing the background, motivation, objectives, and significance of the Sobel edge detection system. A literature review follows, summarizing existing algorithms and FPGA implementations relevant to the project. The methodology chapter explains the design and integration of custom Sobel IP cores within the Vivado environment. Implementation and simulation results are presented next, showcasing the verification of hardware functionality and performance. Subsequent chapters discuss the analysis of timing and resource utilization, as well as the challenges encountered and solutions applied during development. Finally, the report concludes with a summary of achievements and recommendations for future enhancements, supported by references and appendices containing additional technical documentation.

CHAPTER 2

LITERATURE REVIEW

2.1 Implementation of Sobel Edge Detection Using FPGA

Reference

G. Poojitha et al., Proceedings of NC3 2020

Key Concepts

i. FPGA-based implementation of Sobel edge detection using Verilog and Xilinx ISE platform.
ii. Performs gradient calculation with 3x3 convolution kernels to find image edges.
iii. Utilizes MATLAB for image preprocessing and text file generation for simulation input.
iv. Focus on real-time processing of large pixel data using dedicated hardware on FPGA.

Relevance

This study informs the hardware-level design and simulation approach for Sobel filters, validating the use of FPGA for accelerating image edge detection tasks with real-time constraints.

2.2 Efficient Edge Detection on Low-Cost FPGAs

Reference

Jamie Schiel and Andrew Bainbridge-Smith, University of Canterbury

Key Concepts

i. Proposed a new FPGA-optimized Sobel edge detection architecture emphasizing separability and multiplication-free operations.
ii. Introduced custom compressors and look-ahead logic for faster addition and reduced area.
iii. Achieved 28% speed increase and 4.4% area reduction compared to prior FPGA edge detection designs.

Relevance

Offers insights into resource-efficient FPGA architectures, useful for minimizing power and area consumption in embedded edge detection systems like UAVs.

2.3 FPGA-Based Low-Power Sobel Edge Detection System

Reference

Y. Sri Chakrapani et al., IRJMETS, April 2023

Key Concepts

i. Combined Sobel and Canny algorithms implemented on FPGA for enhanced accuracy and robustness.
ii. Included non-maximum suppression and thresholding in post-processing to improve edge quality.
iii. Targeted applications in robotics, surveillance, and autonomous vehicles with real-time performance.

Relevance

This paper demonstrates practical system-level design incorporating edge detection and post-processing on FPGA platforms aligning with project goals of accuracy and real-time execution.

2.4 FPGA Implementation for Real-Time Sobel Edge Detector Using 3-Line Buffers

Reference

Ronnie O. Serfa Juan et al., The IIER International Conference, 2015

Key Concepts

i. Sobel edge detection using a pipelined FSM and 3-line buffering strategy on FPGA to enhance throughput.
ii. Utilized Xilinx Vertex-II FPGA resources effectively with clock synchronization for real-time operation.
iii. Achieved improved Peak Signal to Noise Ratio (PSNR) and clock frequency performance.

Relevance

This implementation provides architectural ideas for line buffering and pipeline synchronization to improve edge detection efficiency on FPGA.

2.5 RESEARCH GAP

a) Implementation of Sobel Edge Detection Using FPGA

focuses on the need for improved synchronization and pipeline control mechanisms tailored for real-time video streaming in FPGA Sobel implementations. Achieving an optimal trade-off between line buffer size, latency, and resource usage also remains an open challenge.

b) Efficient Edge Detection on Low-Cost FPGAs

power consumption remains a critical challenge in deploying FPGA Sobel edge detectors in portable and embedded systems such as UAVs. Current systems often do not balance power, speed, and accuracy effectively, and post-processing stages tend to be offloaded from hardware, affecting system efficiency.

c) FPGA-Based Low-Power Sobel Edge Detection System

highlights the gap in balancing real-time performance with hardware resource utilization and adaptive processing in FPGA-based Sobel edge detection. Many designs overlook integration of threshold adjustments and parallelism to enhance robustness and scalability.

d) FPGA Implementation for Real-Time Sobel Edge Detector Using 3-Line Buffers

points out that existing FPGA edge detection designs do not fully leverage FPGA internal structures like LUTs and fast carry chains, limiting speed and resource efficiency. There is a need for novel compressor-based architectures to optimize both performance and area on low-cost FPGA platforms.

CHAPTER 3

SOFTWARE SETUP

1. Download Vivado

- Go to the official Xilinx (AMD) download page: https://www.xilinx.com/support/download.html
- Sign in or create a free Xilinx account to get access to downloads.
- Choose the appropriate Vivado version Vivado 2018.3.
- Download the Windows Self Extracting Web Installer or a full offline installer, based on your needs and internet connectivity.

2. Install Vivado

- Run the downloaded installer.
- Log in with your Xilinx credentials when prompted.
- Select "Download and Install Now" for a streamlined process.
- Choose Vivado ML Standard (the free version, previously known as WebPack) for most academic and Zynq board projects.
- During setup, ensure you select support for your specific device family (e.g., Zynq-7000).
- Accept the license agreement and specify the installation directory (default is usually fine, e.g., C:\Xilinx).
- Proceed with the installation; this may take up to an hour depending on your internet speed and chosen components.

3.Activate License

- After installation, open the Vivado License Manager (from the start menu or under Vivado's Help menu).
- Select Get Free Licenses and activate the Vivado WebPACK license for non-commercial use. This process is guided within the license manager window.

4. Run and Verify

- Launch Vivado from your desktop or start menu.

- Check that it recognizes your license and that the correct device family (e.g., Zynq, Artix-7) is listed.
- You are now ready to start creating, simulating, and implementing hardware designs for your FPGA project.

This ensures you are running Vivado 2018.3—not a later or earlier version—which is crucial for compatibility with certain IP cores and project files.

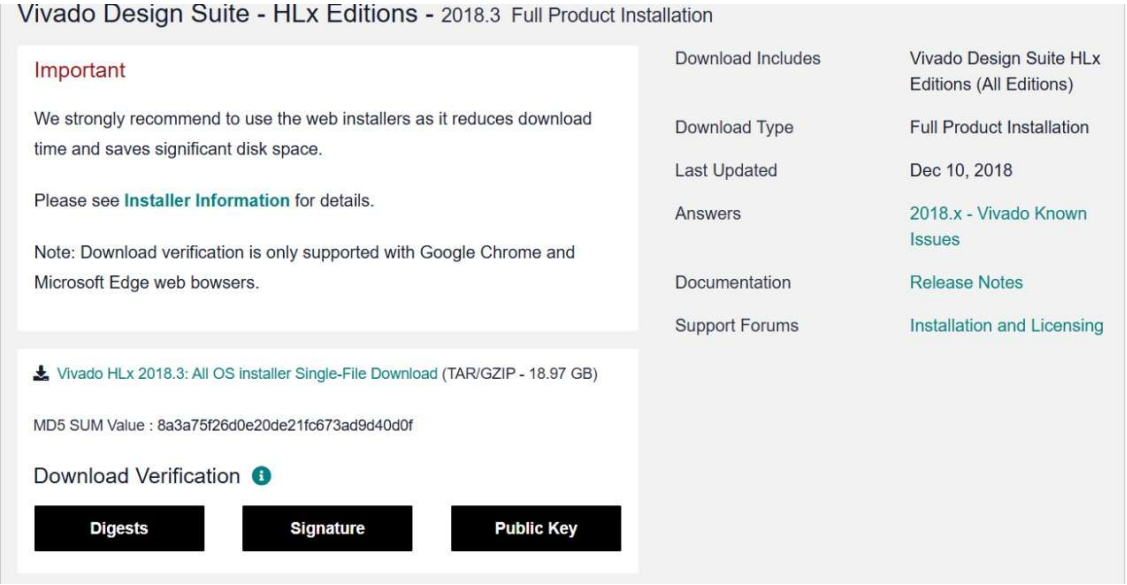

Fig.3.1 Vivado file

CHAPTER 4

MATERIALS AND METHODS

4.1 BLOCK DIAGRAM

The block diagram of the Sobel edge detection system on FPGA typically consists of several key components designed for efficient and real-time image processing. The input image pixels are first stored or buffered, often using Block RAM (BRAM) or line buffers, to facilitate access to neighboring pixel data required for convolution. Custom Sobel IP cores perform the convolution operations separately in the X and Y directions, generating gradient data representing edge information. These gradient outputs are then combined to calculate the edge magnitude. DMA (Direct Memory Access) controllers manage efficient data transfer between the FPGA processing logic and external memory or processing system. The system also incorporates control logic for synchronization, clock, and reset signals. Finally, the output edge-detected image is routed to an output interface such as HDMI or VGA for display. This modular design ensures parallelism, low latency, and efficient resource usage for real-time Sobel edge detection on FPGA platforms.

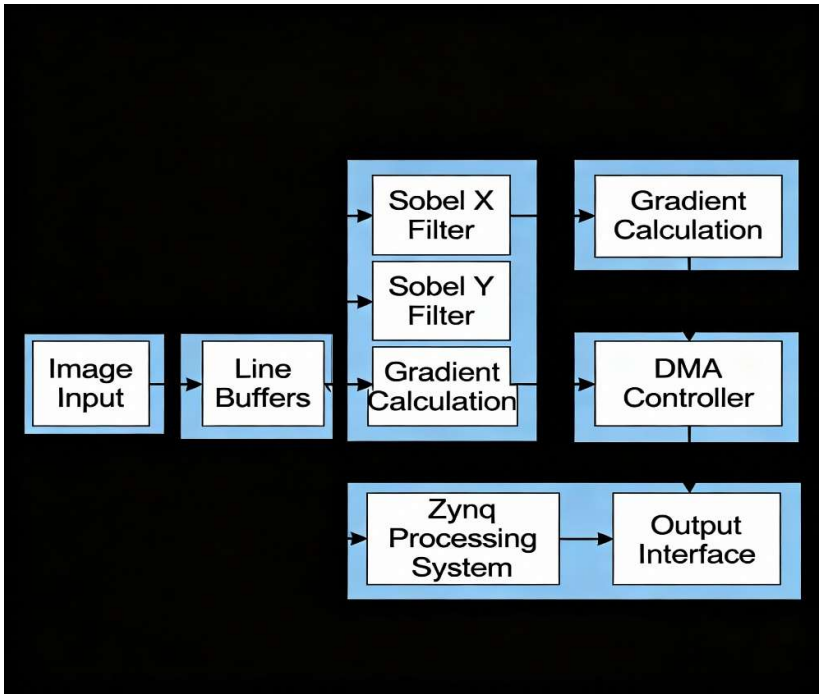

Fig. 4.1 Block Diagram

i. Image Input (from camera or memory) feeds pixels into
ii. Line Buffers/Pixel Buffer, which store neighboring pixels for convolution
iii. Sobel X Filter IP Core and Sobel Y Filter IP Core process pixels in parallel
iv. Gradient Magnitude Calculation combines filter outputs to form edge intensity
v. DMA Controller handles data transfer between FPGA fabric and Processing System
vi. Zynq Processing System manages control and higher-level processing
vii. Output Interface (e.g., HDMI) sends the processed edge-detected image to display.

4.2 IMPLEMENTATION OF THE MODEL

Working of Sobel Filters

1. The Sobel filter is split into two separate hardware IPs—Sobel X and Sobel Y—each performing 1D convolution with 3x3 kernels for horizontal and vertical edge detection.
2. Input pixel data, converted to grayscale, is fed sequentially into line buffers or shift registers in the FPGA to provide the necessary 3x3 pixel window for convolution.
3. Each Sobel IP computes the gradient in its direction by multiplying pixel values by predefined kernel coefficients, performing additions and subtractions accordingly.
4. The outputs from Sobel X and Sobel Y IP cores are transferred via AXI DMA back to memory.
5. Finally, the edge magnitude is calculated by combining the two gradients, typically with an approximate formula like absolute(Gx) + absolute(Gy).

   G = \sqrt{(Sobel_x)^2 + (Sobel_y)^2}

6. Control and data flow between programmable logic (Sobel IPs) and processing system are managed through standardized AXI interfaces, ensuring efficient streaming and synchronization.
7. The entire design, simulated and synthesized in tools such as Vivado, runs on the PYNQ-Z2 board with inputs coming from a webcam and outputs displayed on an HDMI monitor.

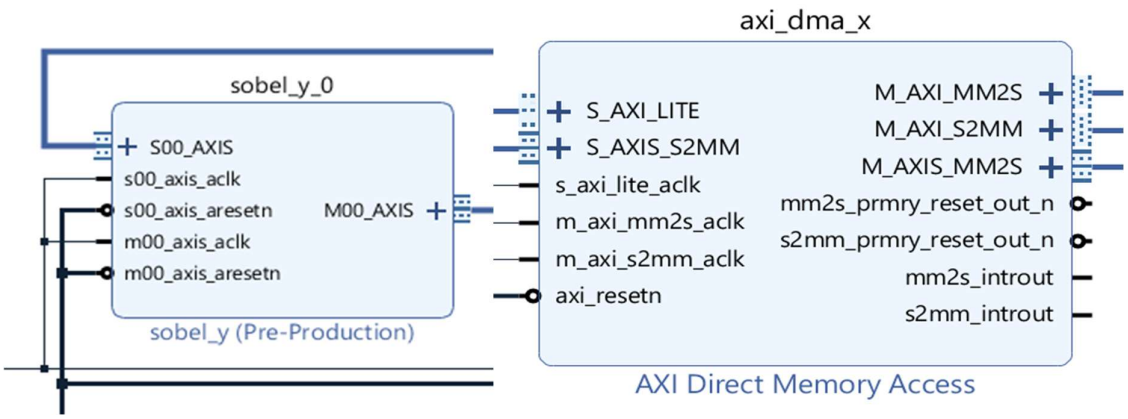

Fig. 4.2 Sobel Filters

4.3 PROCEDURE

a) **Create a New Vivado Project:** Launch Vivado, create an empty RTL project targeting your FPGA board (e.g., ZedBoard or Zybo), select VHDL as the target language, and configure project settings.

b) **Create the Processor System Using IP Integrator:** Use IP Integrator to create a block design. Add the Zynq Processing System (PS) IP and run block automation to configure DDR and Fixed IO connections based on your board.

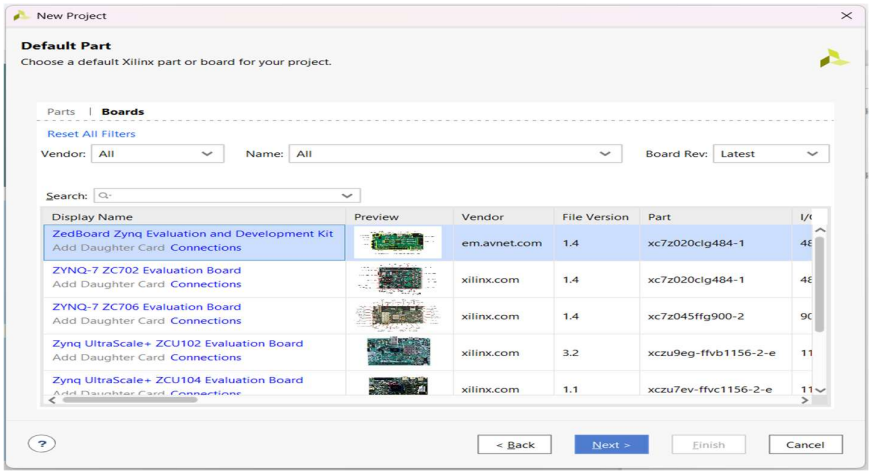

Fig.4.3 Board Selection

c) **Customize Processing System:** Within the PS configuration, enable necessary peripherals like UART for communication and disable unused ones. Setup AXI interfaces and clock/reset options as required.

d) **Add Custom IPs and Interconnect:** Add custom Sobel X and Sobel Y filter IP blocks, connect them to AXI DMA cores and AXI SmartConnect for data streaming, and hook

into the PS via AXI interfaces.

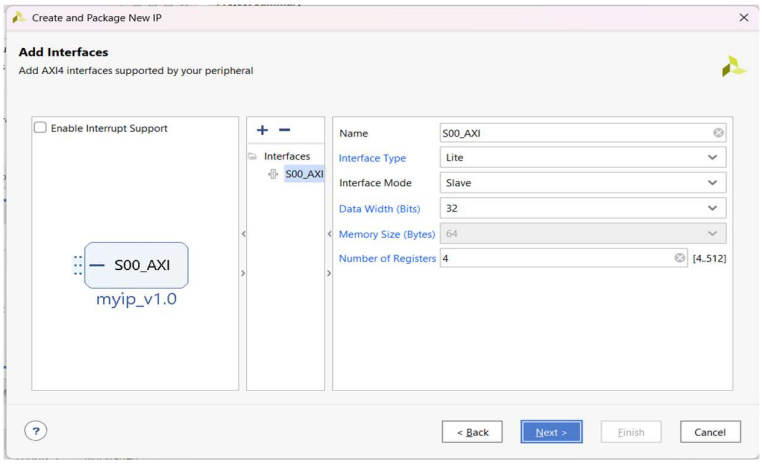

Fig.4.4 Creating IP Blocks

e) **Validate the Design:** Use Vivado's Design Validation to check connectivity and rule compliance.

f) **Generate Output Products and Export to SDK:** Generate HDL wrappers and output products, then export the hardware design to the SDK environment for software development.

g) **Create and Run SDK Application:** In SDK, create an application project (e.g., memory test or edge detection test), build and run it on the board to verify hardware functionality.

h) **Testing in Hardware:** Program the FPGA with your bitstream and execute the software application. Use UART terminal for output logs to confirm operation.

4.4 FLOW CHART

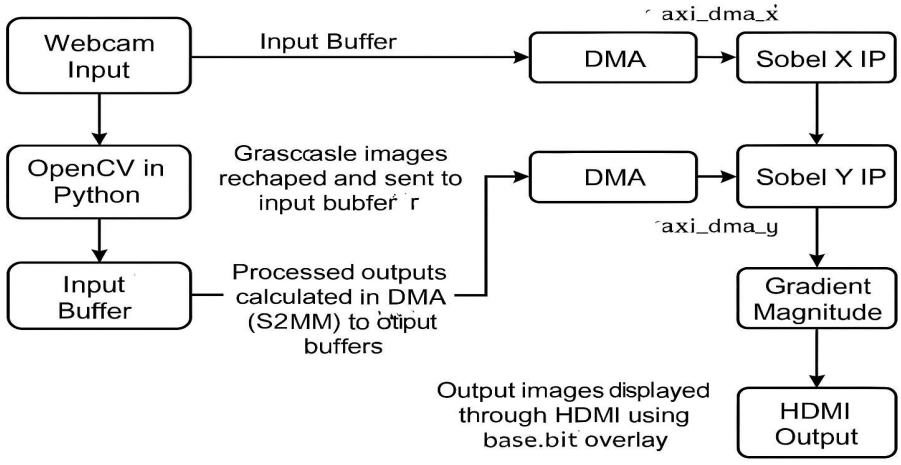

Fig.4.5 Flow Chart

a) **Input Buffer / DMA In:**
    a. Receives pixel data from the processor (via Python/OpenCV) and stores it in memory.
    b. Uses AXI-DMA (Direct Memory Access) to efficiently transfer image frames from the processing system (PS) to programmable logic (PL).
b) **Line Buffer**:
    a. Temporary on-chip memory or FIFO that holds consecutive rows of image data.
    b. Enables the creation of a 3x3 pixel window needed for the Sobel convolution in real time.
c) **Sobel X Filter IP:**
    a. Custom hardware module that applies the horizontal Sobel kernel to each 3x3 window, detecting vertical edges.
    b. Outputs a gradient value (Gx) for each pixel location.
d) **Sobel Y Filter IP:**
    a. Similar custom hardware block that applies the vertical Sobel kernel, detecting horizontal edges.
    b. Outputs a gradient value (Gy) per pixel.
e) **Gradient Magnitude Calculation:**
    a. Receives Gx and Gy outputs from the Sobel filter blocks.
    b. Calculates the total edge strength at each pixel (usually by |Gx| + |Gy|, or using the square root of the sum of squares if resources allow).
f) **Output Buffer / DMA Out:**
    a. Stores processed pixel data for transfer back to main system memory.
    b. Uses another AXI-DMA block to move edge-detected frames from FPGA back to the processing system or directly to the display interface.
g) **Display Output (e.g., HDMI):**
    a. Sends the processed edge-detected image to an HDMI controller.
    b. Allows visualization of results on a connected monitor or screen.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Design of Sobel filters

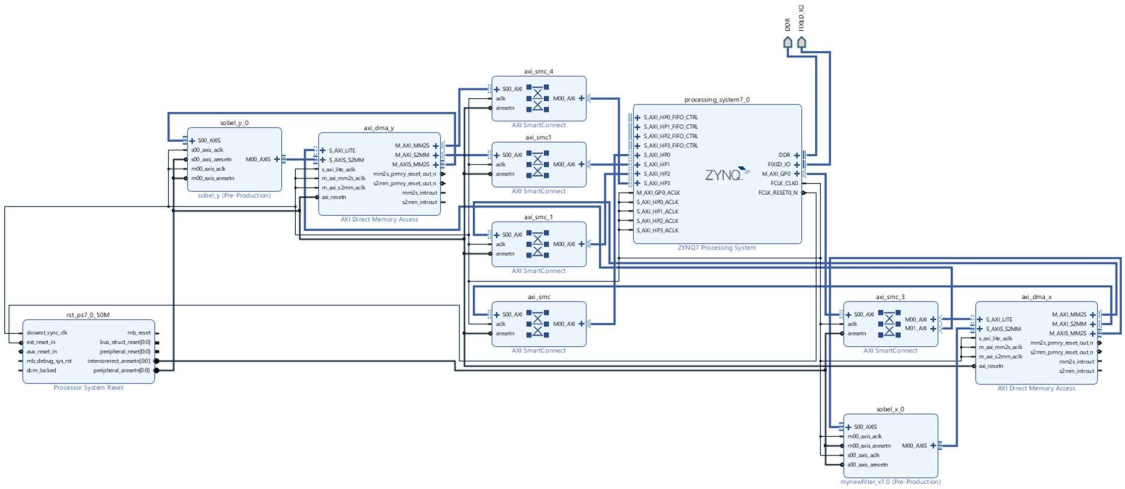

Fig.5.1 Vivado Design

a) By implementing the Sobel edge detection model in Vivado, the key result is a hardware-accelerated system capable of real-time edge detection directly on the FPGA. The Vivado design integrates custom Sobel X and Y IP cores, AXI DMA for fast data transfer, and smart interconnects with the Zynq Processing System. The project demonstrates that the hardware solution is over faster than the equivalent software implementation for Sobel filtering, with hardware processing times around 0.0008 seconds for each filter compared to 0.18 seconds in software.

b) Additionally, post-implementation analysis showed less than 20% overall FPGA resource utilization, 1.7 W on-chip power usage, and successful achievement of timing closure, indicating efficiency and scalability for larger designs. The outputs from hardware—Sobel X, Sobel Y, and gradient computation—closely match the software reference, validating the correctness of the hardware filter IPs. The processed images are displayed in real time on an HDMI monitor, confirming the design accomplishes real-time edge processing as intended.

c) Though the hardware portion (Sobel X and Y) is extremely fast, the gradient magnitude calculation (still performed in software) limits the total acceleration. Future improvements would include migrating more processing (like gradient and thresholding) into hardware to fully leverage the speedup potential of the FPGA.

5.2 Output of Sobel Filters in FPGA

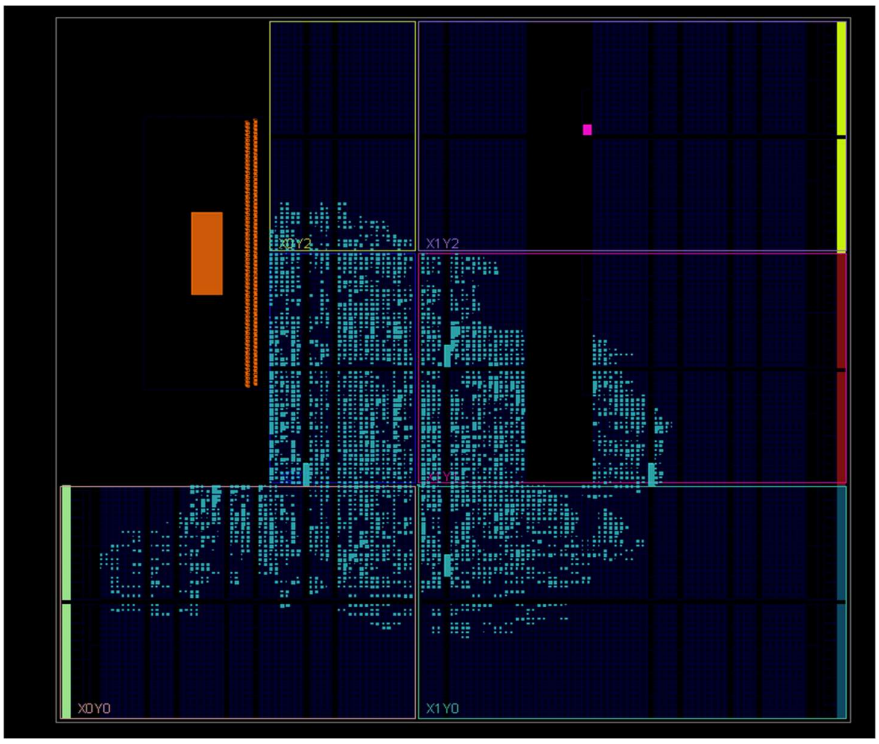

Fig.5.2 Output of Sobel Filters

The image obtained after running implementation in Vivado is a "device utilization map" or **FPGA resource placement view**. This view shows how the Vivado tool mapped your Sobel edge detection hardware design onto the physical regions of the FPGA chip.

i. The coloured blocks represent areas used for logic (LUTs), flip-flops, DSP slices, and other FPGA resources consumed by your custom modules, DMA, and processing system interconnects.

ii. Most activity clusters in the central region, indicating that Sobel X and Y IP cores, memory interfaces, and control logic are efficiently packed.

iii. The sparse or unmarked regions show unused FPGA fabric, confirming your design uses only a fraction of available resources, which is typical for a proof-of-concept or phase 1 block-level implementation.

iv. The clear structure and lack of congestion demonstrate a successful place-and-route process, suggesting your block design achieved timing closure and is ready for hardware programming and further testing.

In summary, this result validates that Vivado correctly mapped your block design onto the chip, and your system is efficient and scalable for either more complex image processing.

CHAPTER 6

PHASE II OBJECTIVES & SCHEDULE

Phase II Objectives

- Migrate the gradient magnitude calculation from software (Python) to hardware by developing a dedicated IP core for this operation, thereby enabling fully hardware-accelerated edge detection.
- Integrate post-processing steps such as thresholding or binarization into hardware to improve the clarity of edge maps and overall system speed.
- Modify the Vivado block design to incorporate HDMI or real-time video output directly in the custom overlay, eliminating the need to switch overlays and making the output pipeline more efficient.
- Optimize resource utilization and further reduce latency by refining AXI interconnects and memory mapping for high-throughput image streaming.
- Validate hardware outputs by comparing them against both the previous software solution and real-time video input cases.
- Develop test infrastructure for continuous streaming and edge visualization.

Phase II Schedule

- Weeks 1–2: Design and simulation of hardware gradient magnitude and thresholding IP cores.
- Weeks 3–4: Integration of new IPs into Vivado block design, updating DMA and HDMI modules.
- Weeks 5–6: Synthesis, implementation, bitstream generation; initial debugging and functional verification on hardware.
- Weeks 7–8: System-level integration, testing with various input sources (webcam, video streams), and quality validation.
- Weeks 9–10: Performance benchmarking, resource and timing optimization, documentation of results, and report preparation.

REFERENCES

1. Schiel, J., & Bainbridge-Smith, A. "Efficient Edge Detection on Low-Cost FPGAs." Department of Mechatronic Engineering, University of Canterbury, Christchurch, New Zealand.
2. Poojitha, G., Sri Chakrapani, Y., & Venkateswara Rao, N. "Implementation of Sobel Edge Detection Using FPGA." Proceedings of NC3 2020, Bapatla Engineering College, pp. 94–99, 2020.
3. Sri Chakrapani, Y., Rohita Sai, A., Jo Sri Ganesh, B., Pawan Kalyan, B., & Swathi, E. "FPGA-Based Low-Power Sobel Edge Detection System." International Research Journal of Modernization in Engineering, Technology and Science, vol. 5, no. 4, pp. 3039–3045, April 2023.
4. Serfa Juan, R. O., Park, C. S., Kim, H. S., & Cha, H. W. "FPGA Implementation for Real-Time Sobel Edge Detector Block Using 3-Line Buffers." Proceedings of 25th TheIIER International Conference, Kuala Lumpur, Malaysia, pp. 58–63, May 2015.

ANEXURE

A. Synthesis and Implementation Reports

a. Summary table of FPGA utilization (LUT, FF, BRAM, DSP) for the implemented design.
b. Timing analysis and worst negative slack after implementation.
c. Power analysis report (on-chip power consumption, estimated temperature).

B. Sample Code Snippets

a. Top-level VHDL/Verilog module header for the custom Sobel IP core.
b. Example Python code (Jupyter cell) interfacing with FPGA hardware for image acquisition and result visualization.

C. Testbench Setup

a. Block diagram or text description of the test environment (Webcam input, DMA, FPGA processing, HDMI output).
b. Reference to simulation waveforms and explanation of signal meanings.

D. Experimental Images and Outputs

a. Example input images used for validation.
b. Edge-detected output images (from both software and hardware) for reference.

E. Additional Documentation

a. List of Vivado and board support files used.
b. License information or relevant third-party acknowledgements.
c. Troubleshooting notes or frequently encountered FPGA setup issues.