

**Bharat AI-SoC Student Challenge**  
**Project Report**

**Problem statement 5:**  
**Real Time Object Detection Using Hardware Accelerated CNN**  
**on Xilinx Zynq FPGA with ARM Processor**

*Submitted by Anika Jha*

# INDEX

1. Abstract
2. Introduction
3. Objectives
4. Methodology
5. Result Analysis
6. Conclusion and Summary

# **ABSTRACT**

Object detection has become a fundamental component of modern intelligent systems, particularly in edge applications such as surveillance, robotics, autonomous monitoring, and smart embedded devices. However, deploying deep learning models like YOLO on embedded platforms presents significant computational challenges due to limited processing power and energy constraints. This project addresses these challenges by implementing and optimizing a YOLO-based object detection system using a Xilinx Zynq FPGA platform to achieve accelerated inference performance.

The proposed system adopts a hardware software co-design approach in which computationally intensive convolution operations are offloaded to the FPGA programmable logic, while preprocessing, control flow, and post-processing are executed on the ARM processor. The YOLO model is implemented in Python and integrated with custom hardware accelerators designed using Vivado. Optimization techniques such as loop pipelining, loop unrolling, and parallel processing are applied to improve throughput and reduce latency.

Performance evaluation demonstrates that the hardware-accelerated implementation provides a speedup compared to the CPU-only execution while maintaining detection accuracy. The system also shows improved power efficiency and effective utilization of FPGA resources. This work highlights the use of Zynq architecture for real-time edge AI applications and demonstrates practical acceleration of convolutional neural networks in embedded environments.

# INTRODUCTION

The rapid advancement of artificial intelligence and deep learning has significantly improved computer vision systems, especially in the area of object detection. Object detection allows a system to identify and locate multiple objects within an image simultaneously, making it essential for applications such as surveillance systems, traffic monitoring, smart robotics, and embedded edge devices. Among various detection algorithms, YOLO is widely recognized for its ability to perform detection in a single forward pass, enabling faster inference compared to traditional region-based approaches. This makes it suitable for real-time implementations.

However, running YOLO entirely on an embedded ARM processor presents performance limitations. Convolutional neural networks require intensive multiply-accumulate operations, especially in convolution layers. When executed only on the ARM cores of a Zynq device, inference latency increases significantly, reducing throughput and limiting real-time performance. To address this limitation, hardware acceleration using the programmable logic of the Zynq FPGA was implemented in this project.

In this work, YOLO was developed and tested using Python in Visual Studio Code for flexibility and ease of integration. A custom CNN accelerator IP core was designed and connected using AXI interfaces to offload convolution computations to the FPGA fabric. The ARM processor manages image preprocessing, data transfer, and post-processing, while the CNN accelerator performs parallel convolution operations. This hardware software co-design approach improves performance and demonstrates efficient utilization of the Zynq heterogeneous architecture for embedded object detection.

## **OBJECTIVES**

The primary objective of this project is to design and implement an FPGA-accelerated object detection system using the YOLO algorithm on a Xilinx Zynq platform. The goal is to enhance inference performance by offloading computationally intensive convolution operations from the ARM processor to the programmable logic using a custom CNN accelerator connected through AXI interfaces.

A key objective is to achieve measurable performance improvement compared to a CPU-only implementation of YOLO running in Python. This includes reducing inference latency, increasing throughput, and achieving at least two times speedup over the ARM-based execution. Another important objective is to ensure that acceleration does not compromise detection accuracy while maintaining stable system operation.

The project also aims to efficiently utilize FPGA resources such as LUTs, DSP slices, and BRAM while meeting timing constraints. Optimizing data movement between processing system and programmable logic through AXI communication is a core design focus.

Additionally, the project seeks to demonstrate effective hardware software co-design by partitioning tasks appropriately between software running on the ARM processor and hardware implemented in programmable logic. Preprocessing, control flow, and bounding box generation are handled in Python, while convolution layers are accelerated in hardware.

Finally, the project aims to provide practical understanding of embedded AI deployment, FPGA-based acceleration, and performance optimization techniques in heterogeneous systems.

## METHODOLOGY

This project focuses on accelerating object detection using the YOLO algorithm by leveraging the hardware capabilities of a Xilinx Zynq FPGA platform. The main objective was to improve inference performance compared to a CPU-only implementation in Python. YOLO was selected because it performs detection in a single forward pass, making it suitable for real-time applications. However, even lightweight YOLO variants can be computationally intensive when executed purely on an embedded ARM processor.

The development followed a hardware software co-design methodology. First, the YOLO model was implemented and tested in Python to establish baseline accuracy and performance metrics. Profiling tools were used to identify computational bottlenecks, particularly in convolution layers and matrix multiplications. These operations were then targeted for hardware acceleration. The system was partitioned so that preprocessing, control logic, and post-processing ran on the ARM processor, while convolution-intensive computations were offloaded to the FPGA fabric.

The accelerator was developed using Vivado and integrated using AXI interfaces for communication between processing system and programmable logic. Iterative optimization techniques such as pipelining, loop unrolling, and memory partitioning were applied to increase throughput and reduce latency.

**Hardware Architecture and Platform:** The implementation was deployed on a Xilinx Zynq-based development board, which integrates ARM Cortex-A processors with FPGA programmable logic on a single chip. This heterogeneous architecture enables tight coupling between software flexibility and hardware acceleration. The programmable logic was utilized to implement custom convolution accelerator modules, while the ARM processor handled operating system tasks, image capture, and overall system control.

The object detection pipeline begins with image acquisition through a USB camera or stored dataset. Images are preprocessed using Python and OpenCV on the ARM processor. Preprocessed data is transferred to the FPGA via AXI interfaces.

The hardware design was created in Vivado, where custom IP blocks were connected using AXI interconnects. Resource utilization was carefully monitored to ensure efficient use of LUTs, BRAMs, and DSP slices. Timing closure and clock optimization were also considered to maximize operating frequency.

**Software Design:** The software component of the project was implemented primarily in Python for flexibility and rapid prototyping. The YOLO model was integrated using deep learning frameworks compatible with embedded deployment. Python handled tasks such as image capture, resizing, normalization, and post-processing including bounding box generation and confidence thresholding.

To integrate hardware acceleration, a communication layer was developed between Python and the FPGA accelerator. This layer used memory-mapped interfaces and device drivers to trigger hardware execution and retrieve computed feature maps. The ARM processor coordinated data movement between DDR memory and programmable logic using AXI master interfaces.

OpenCV was used extensively for image preprocessing and visualization of detection results. The system allowed real-time display of bounding boxes around detected objects. Performance measurements were captured for both CPU-only execution and hardware-accelerated execution. The integration demonstrated effective cooperation between software running on ARM cores and hardware logic implemented in FPGA fabric, showcasing a practical hardware software co-design approach for embedded AI systems.

## RESULT ANALYSIS

The final deliverable of this project is a working FPGA-accelerated object detection system using YOLO running on a Zynq platform. The system demonstrates successful hardware software co-design, where computationally heavy CNN layers are implemented in programmable logic while control and preprocessing tasks run on the ARM processor. A comparative analysis between CPU-only and hardware-accelerated implementations highlights improvements in latency, throughput, and efficiency. This project provided valuable insight into embedded edge AI deployment. It strengthened understanding of FPGA-based acceleration, AXI interfacing, and heterogeneous system design. Performance optimization was a central focus of this project. The baseline CPU-only YOLO implementation was profiled to measure inference latency and throughput. The major bottleneck was identified in convolution layers, which involve intensive multiply-accumulate operations. By offloading these computations to the FPGA, substantial performance improvements were achieved.

In hardware, loop pipelining allowed multiple operations to execute concurrently, significantly reducing clock cycles per convolution. Loop unrolling increased parallelism by replicating computation units. Memory partitioning enabled simultaneous access to multiple data elements, preventing memory access from becoming a bottleneck. DSP slices were used efficiently to accelerate arithmetic operations, while BRAM blocks reduced dependency on slower external memory.

After hardware acceleration, inference latency was reduced considerably compared to CPU-only execution. The system achieved more than two times speedup while maintaining detection accuracy. Throughput improved to near real-time performance depending on input resolution. Additionally, energy efficiency per inference improved due to parallel hardware execution. Resource utilization reports confirmed balanced usage of FPGA components without exceeding device constraints.

Overall, the project demonstrates how combining FPGA acceleration with software-based deep learning frameworks can significantly enhance object detection performance in embedded systems, making it suitable for real-world edge applications such as surveillance, robotics, and smart monitoring systems.

## **SUMMARY AND CONCLUSION**

This project successfully demonstrates the implementation of an FPGA-accelerated object detection system using the YOLO algorithm on a Xilinx Zynq platform. The system was developed using Python in Visual Studio Code for model integration, preprocessing, and post-processing, while a custom CNN accelerator IP core was designed in the FPGA programmable logic to accelerate convolution operations. Communication between the ARM processing system and the programmable logic was achieved using AXI interfaces, enabling efficient data transfer and hardware control.

The initial CPU-only implementation of YOLO on the ARM processor was profiled to identify computational bottlenecks, particularly in convolution layers involving intensive multiply-accumulate operations. These operations were offloaded to the CNN accelerator to exploit hardware-level parallelism. By leveraging FPGA resources such as DSP slices for arithmetic operations and BRAM for intermediate data storage, the system achieved significant performance improvements.

Performance comparison showed clear reduction in inference latency and improved throughput compared to the software-only implementation. The design also demonstrated efficient FPGA resource utilization while maintaining detection accuracy. Overall, the project highlights the benefits of heterogeneous computing using the Zynq architecture, combining software flexibility with hardware acceleration for embedded AI applications.