REAL-TIME OBJECT DETECTION USING HARDWARE ACCELERATED CNN ON XILINX ZYNQ FPGA WITH ARM PROCESSOR

FINAL PROJECT REPORT

Submitted by

M.V. ABISHEK

D. GIRITHARAN

S. NISANTH

DECLARATION

We hereby declare that the project entitled “Real-Time Object Detection Using Hardware Accelerated CNN on Xilinx Zynq FPGA with Arm Processor” is an original work carried out by us under the guidance of our supervisor. This work has not been submitted, either in whole or in part, to any other university or institution for the award of any degree or diploma. All sources of information used in this project have been duly acknowledged

ABSTRACT

Problem Statement

Real-time object detection and image classification using convolutional neural networks (CNNs) are computationally intensive and difficult to achieve on embedded systems using CPU only processing. The goal is to achieve real-time performance with improvement in latency and throughput compared to a software only implementation, while maintaining efficient FPGA resource and power usage.

Proposed Solution

This project proposes a hardware accelerated CNN inference system using a Xilinx ZedBoard (Zynq-7000 SoC). Images are provided through a stored dataset, and a custom three layer lightweight CNN is designed for efficient execution. The Arm processor handles data loading and control, while the FPGA fabric accelerates convolution. The hardware accelerated design is compared with a CPU only implementation to demonstrate improved inference speed and efficiency.

Keywords

FPGA acceleration, Convolutional Neural Network (CNN), Xilinx Zynq, ZedBoard, Hardware–Software Co-Design, Edge AI, Embedded Systems, Vitis HLS

LIST OF SYMBOLS AND ABBREVIATIONS

FPGA	-	Field Programmable Gate Array
CNN	-	Convolution Neural Networks
AXI	-	Advanced eXtensible Interface
HDL	-	Hardware Description Language
PS	-	Processing System
PL	-	Programmable Logic
HLS	-	High Level Synthesis
UART	-	Universal Asynchronous Receiver Transmitter
LUT	-	Look Up Table
DSP	-	Digital Signal Processing
BRAM	-	Block Random Access Memory

CHAPTER 1: INTRODUCTION

1.1 Need for Hardware Acceleration

CNN inference requires a large number of multiply-accumulate operations, making software-only execution on embedded processors slow and power inefficient. As the complexity of CNN models increases, CPU-based implementations struggle to meet real-time performance requirements, especially in edge applications.

Hardware acceleration using FPGA fabric provides massive parallelism, pipelining, and dedicated arithmetic resources such as DSP blocks and BRAMs. By offloading compute-intensive operations like convolution, activation, and pooling to hardware, the system can achieve significantly lower latency and higher throughput.

The need for hardware acceleration is therefore essential to:

- Achieve real-time or near real-time CNN inference
- Reduce execution latency and power consumption
- Improve system efficiency on embedded platform

1.2 Problem Definition

Real-time CNN inference on embedded systems using CPU-only execution suffers from high latency and limited throughput due to computational constraints. Although GPUs can accelerate CNNs, they are often unsuitable for embedded edge systems due to higher power consumption and cost.

The problem addressed in this project is to design and implement an efficient CNN inference system on an embedded platform that overcomes the limitations of software-only execution. The challenge lies in effectively partitioning the workload between the processor and FPGA fabric, designing a lightweight CNN suitable for hardware implementation, and quantitatively demonstrating performance improvements over a CPU-only approach.

1.3 Project Objectives

The objectives of this project are to design and implement a custom three-layer CNN on a Xilinx ZedBoard (Zynq-7000 SoC) and accelerate compute-intensive operations using FPGA programmable logic. The system performs image inference using a pre-stored dataset and evaluates performance by comparing CPU-only and hardware-accelerated implementations in terms of latency, throughput, and resource utilization.

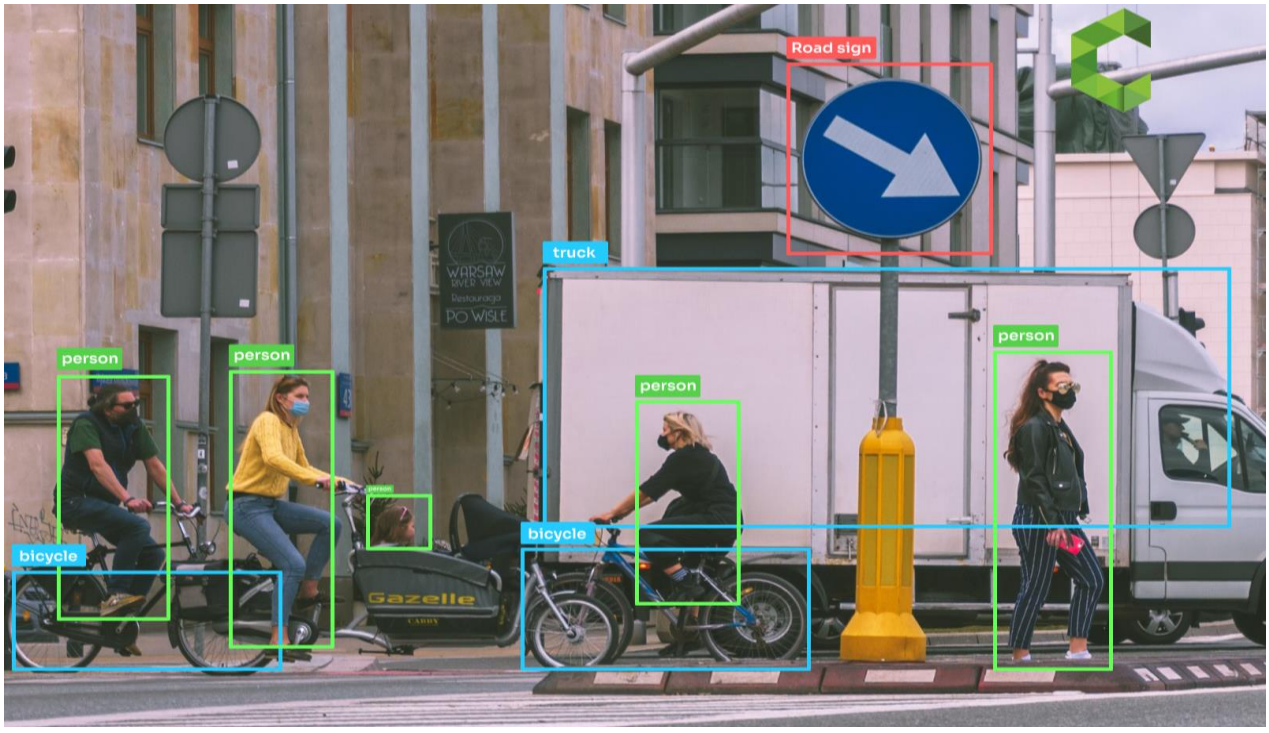
1.4 Scope of the Project

The scope of this project is limited to CNN inference and does not include CNN training on hardware. The system uses dataset-based image input instead of live camera streaming to ensure consistent and repeatable performance evaluation.

The project focuses on:

- Hardware/software co-design using Arm processor and FPGA fabric
- Implementation of a custom three-layer CNN
- Performance evaluation on the ZedBoard platform

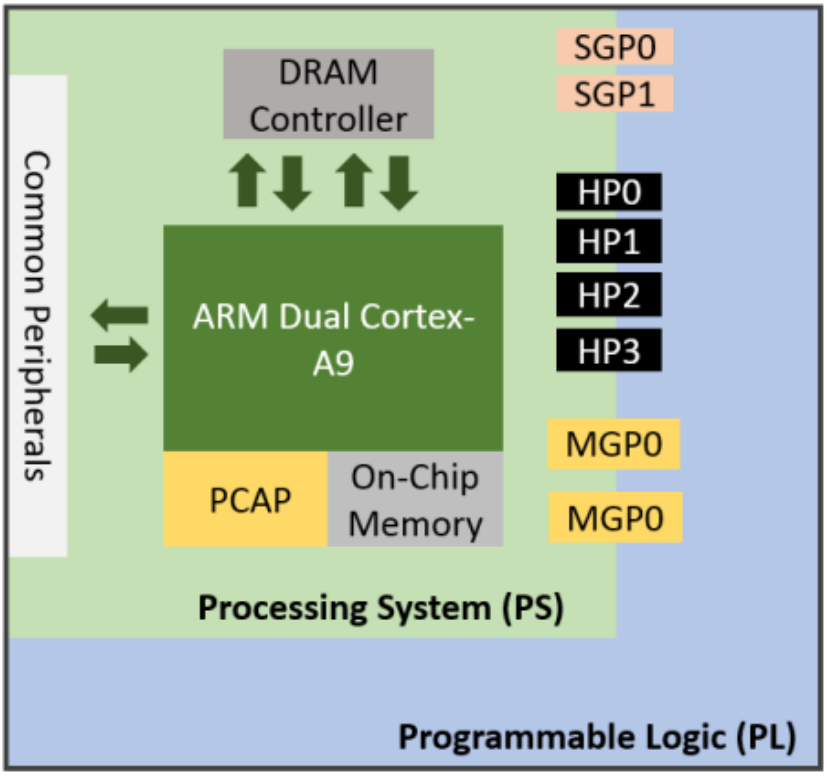
Advanced CNN models, multi-camera support, and cloud integration are outside the scope of this work. However, the proposed design provides a strong foundation for extending the system to more complex models and real-time video processing in the future



CHAPTER 2: SYSTEM OVERVIEW

2.1 Overall System Architecture

The proposed system is implemented on a Xilinx ZedBoard (Zynq-7000 SoC), which integrates an Arm Cortex-A9 Processing System (PS) with FPGA based Programmable Logic (PL). Input images are provided through a pre-stored dataset and processed by a custom three-layer CNN. The system architecture divides tasks between software and hardware to achieve efficient CNN inference with improved performance.



2.2 Hardware-Software Co-Design Concept

The design follows a hardware software co-design approach. The Processing System (PS) handles dataset loading, image preprocessing, control logic, and result display. The Programmable Logic (PL) accelerates compute-intensive CNN operations such as convolution and activation. Data transfer between PS and PL is performed using AXI and DMA interfaces, ensuring high-speed communication and minimal processor overhead.

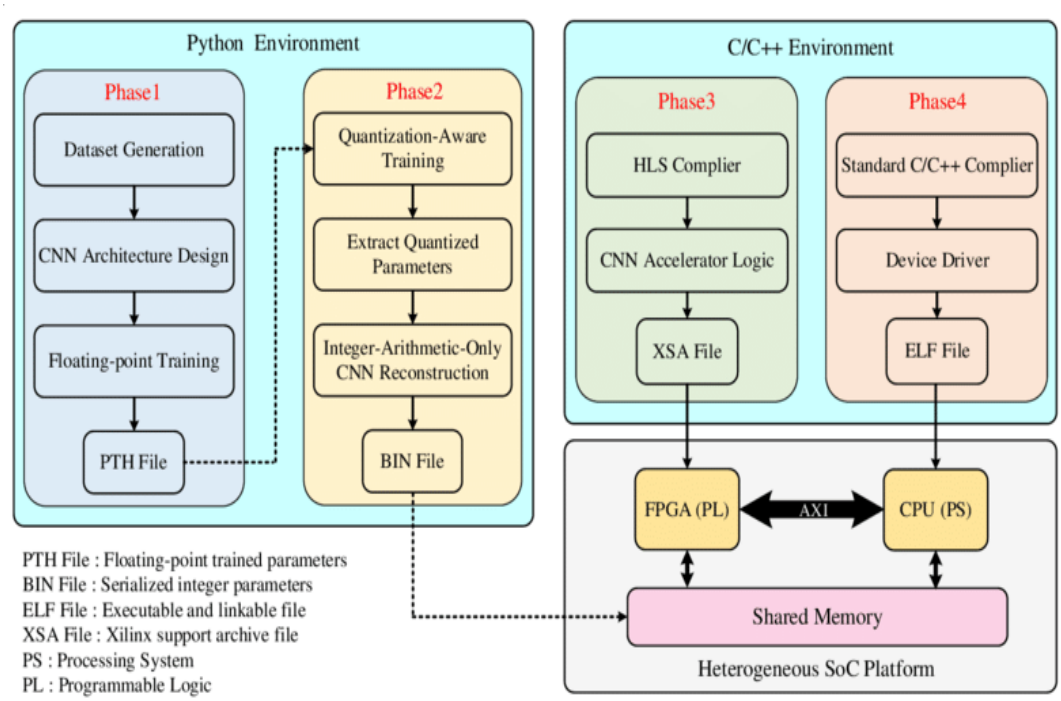
2.3 Design Flow Overview (Phase 1 to Phase 4)

The overall design flow is divided into four phases:

- **Phase 1:** CNN model design and software implementation on the Arm processor
- **Phase 2:** Identification of compute intensive CNN layers for hardware acceleration
- **Phase 3:** Hardware accelerator implementation using Vitis HLS / Vivado
- **Phase 4:** System integration, testing, and performance evaluation

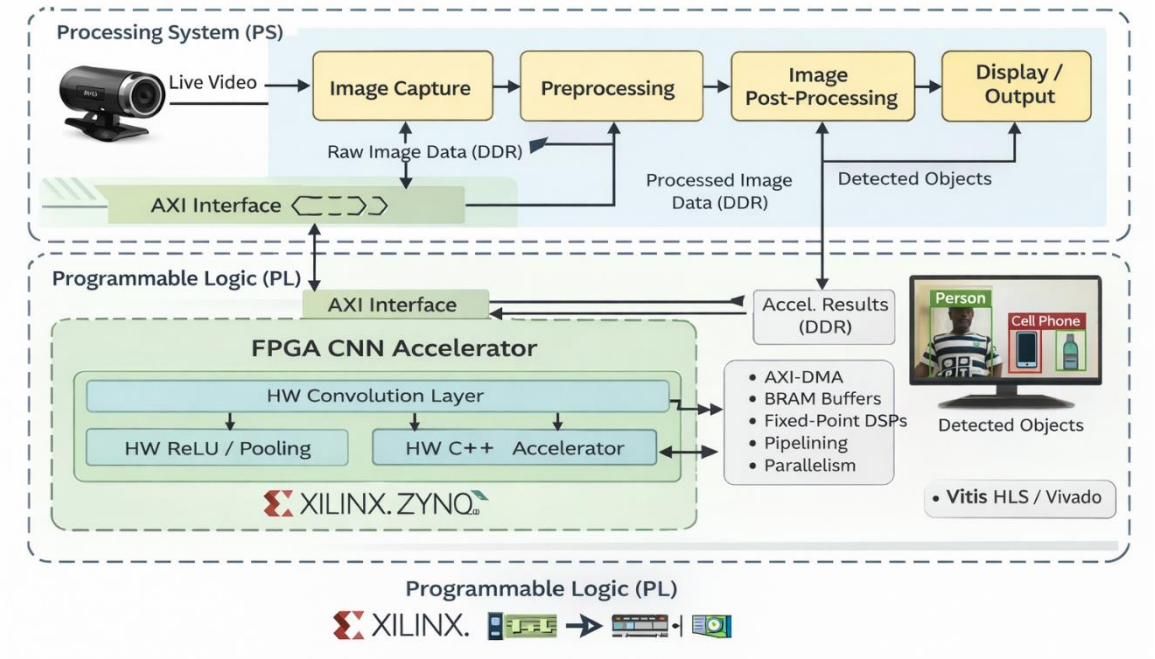
This structured flow enables systematic development and performance optimization.

The following images depict the end-to-end design flow, starting from CNN model development to final system integration and performance evaluation. This phased approach ensures systematic development, easier debugging, and effective performance optimization.



2.4 Block Diagram Description

The system block diagram consists of the dataset input, preprocessing unit in the PS, CNN accelerator in the PL, memory interfaces, and output display. The PS controls data flow and system execution, while the PL performs parallel CNN computations. AXI interconnects and enable efficient data movement between memory and the accelerator, resulting in reduced latency and higher throughput.

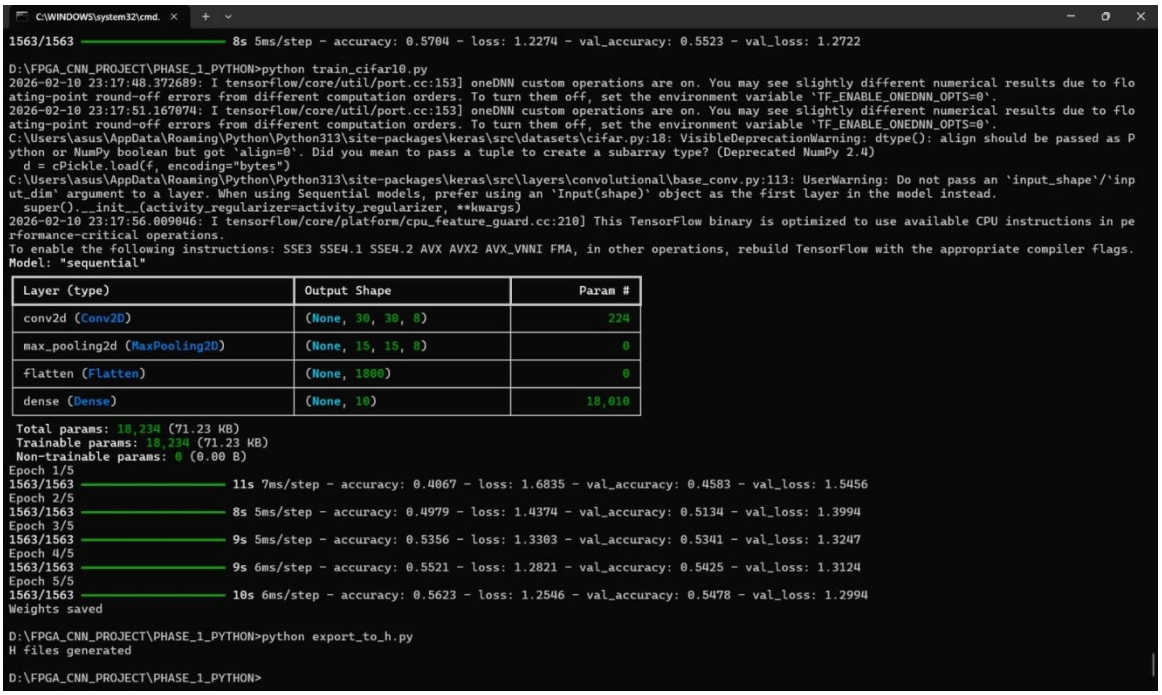


This block diagram illustrates a hardware–software co-designed real-time object detection system implemented on a Xilinx Zynq SoC. The ARM Processing System (PS) handles image capture, preprocessing, post-processing, and display, while the FPGA Programmable Logic (PL) accelerates compute-intensive CNN operations such as convolution, ReLU, and pooling. High-speed data transfer between PS and PL is achieved using AXI and AXI-DMA interfaces with shared DDR memory.

CHAPTER 3: METHODOLOGY

3.1 Phase 1 – CNN Training

A lightweight custom three-layer CNN is designed and trained using Python-based deep learning frameworks. The model is trained on a standard image dataset and optimized for inference on embedded hardware. After training, the model parameters such as weights and biases are extracted and formatted for hardware implementation.

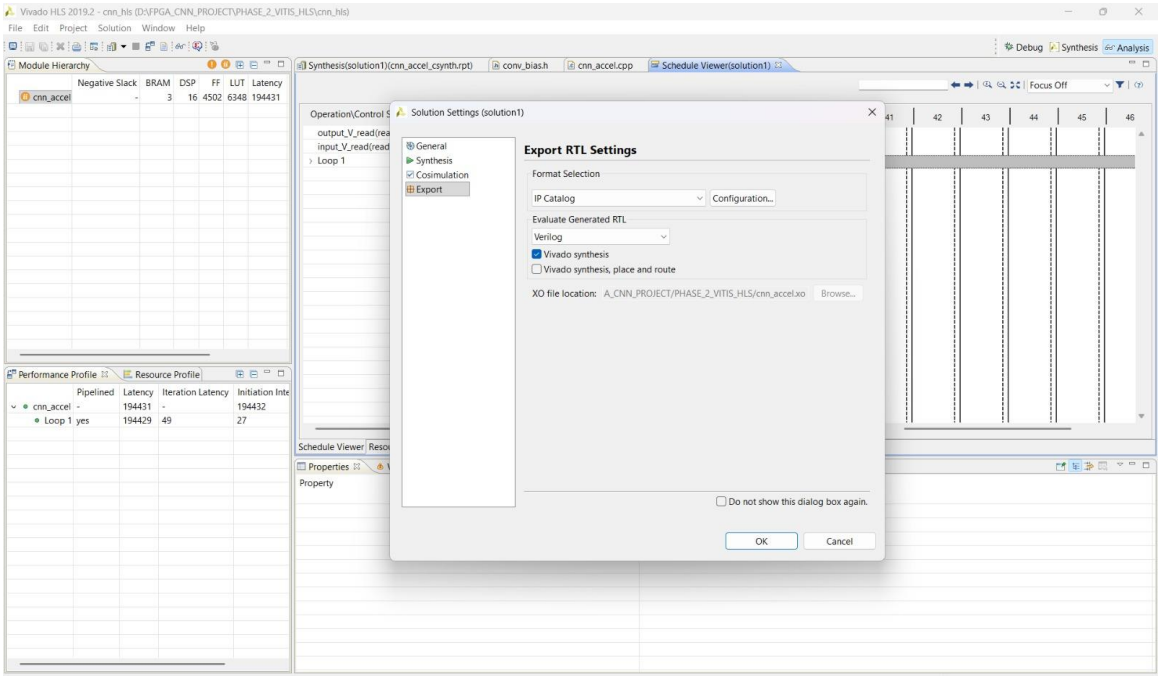


3.2 Phase 2 – HLS IP Generation

The compute-intensive CNN layers, mainly convolution and activation, are implemented as hardware accelerators using Vitis HLS. The CNN operations are described in C/C++ and optimized using loop unrolling, pipelining, and fixed-point arithmetic to improve performance and reduce resource usage. The HLS design is synthesized to generate a reusable IP core.

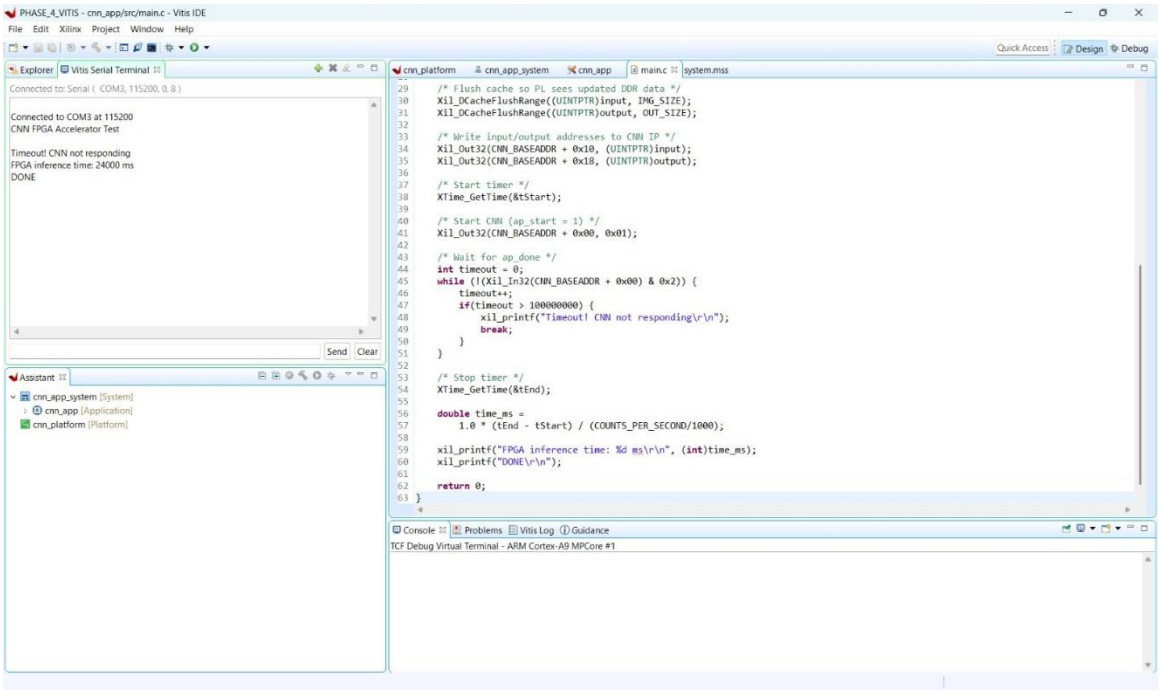
3.3 Phase 3 – Vivado Integration

The generated HLS IP core is integrated into the Zynq system using Vivado Design Suite. AXI interfaces are used to connect the accelerator with the Processing System and memory. The complete hardware design is synthesized, implemented, and a bitstream is generated for the ZedBoard.



3.4 Phase 4 – Vitis Software Development

Embedded software is developed using Vitis to control the hardware accelerator. The Arm processor loads images from the dataset, performs preprocessing, transfers data to the accelerator using DMA, and collects inference results. The application also handles performance measurement and output display.



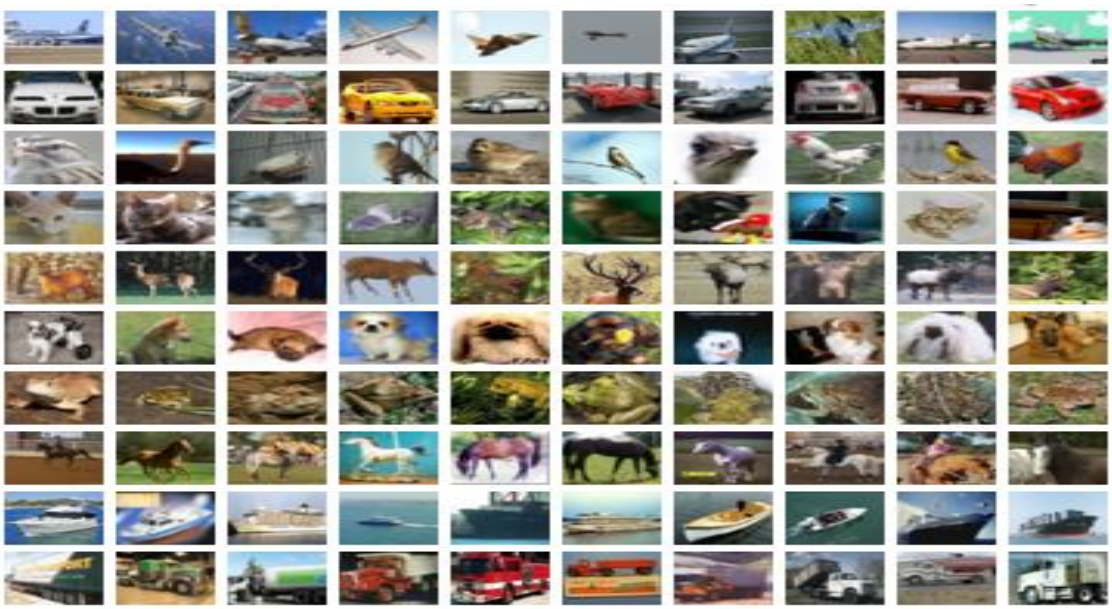
3.5 Final Testing and Validation Strategy

The system is tested using a pre-stored image dataset to ensure consistent evaluation. Functional correctness is verified by comparing hardware-accelerated results with CPU-only inference outputs. Performance is evaluated based on latency, throughput, and resource utilization to validate the effectiveness of hardware acceleration.

CHAPTER 4: CNN MODEL DESIGN

4.1 Dataset Description (CIFAR-10)

The CIFAR-10 dataset is a standard image classification dataset consisting of 60,000 RGB images of size 32×32 pixels distributed across 10 classes. It provides a balanced dataset suitable for evaluating lightweight CNN models. The dataset is divided into training and testing sets to ensure proper validation. Its small image size makes it ideal for embedded and FPGA-based inference.



4.2 Model Architecture

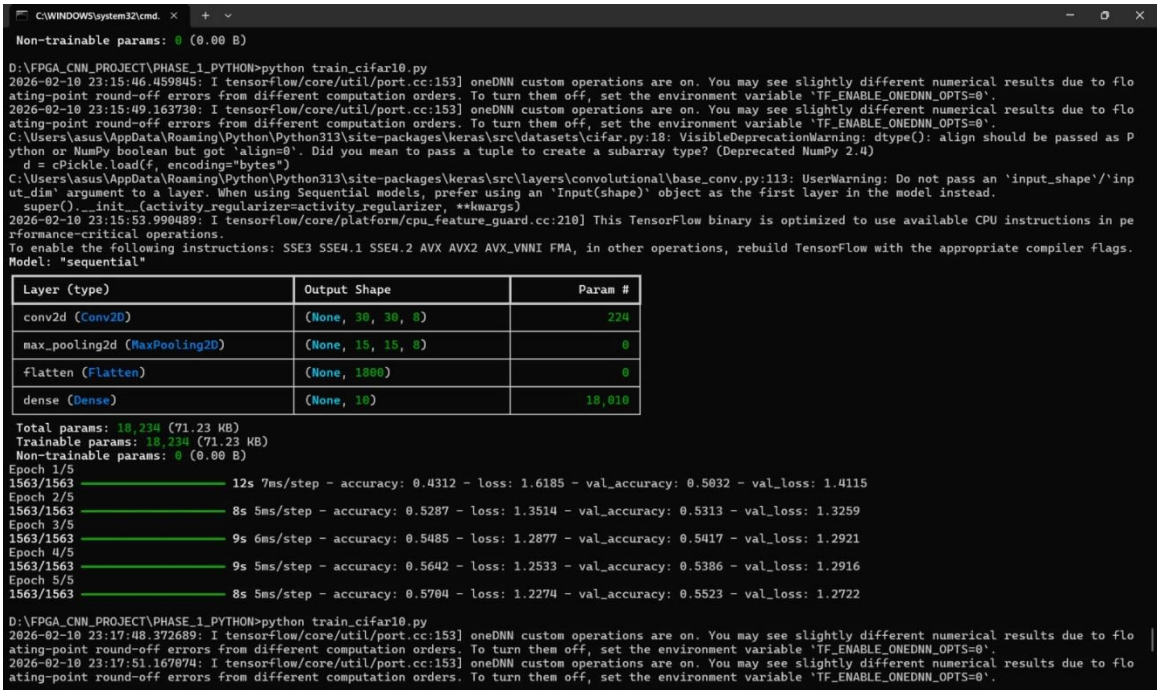
A custom three-layer convolutional neural network is designed to balance classification accuracy and hardware efficiency. The architecture includes convolution, activation, pooling, and a fully connected output layer. The model is kept lightweight to reduce computational complexity. This design ensures suitability for FPGA implementation.

4.3 Training Configuration

The CNN is trained using Python-based frameworks with standard optimization techniques. Training parameters such as learning rate, batch size, and number of epochs are selected to achieve stable convergence. The model is trained offline on a workstation. Only inference is performed on the embedded platform.

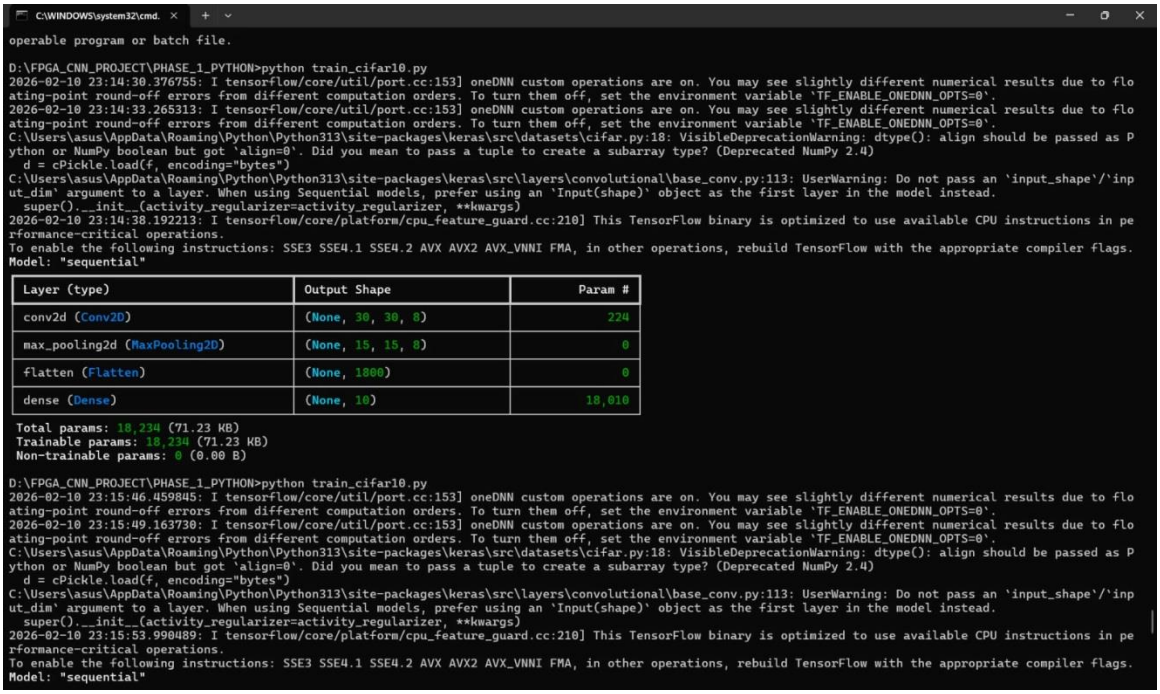
4.4 Accuracy Results

The trained CNN achieves consistent classification accuracy on the CIFAR-10 test dataset. The accuracy validates the correctness of the model architecture and training process. Minor accuracy degradation is acceptable due to hardware-oriented optimizations. The results confirm suitability for embedded inference.



4.5 Weight Extraction and Fixed-Point Conversion

After training, floating-point weights are extracted from the model. These weights are converted to fixed-point representation to reduce FPGA resource usage. Quantization is carefully performed to minimize accuracy loss. The converted weights are used in the hardware accelerator.



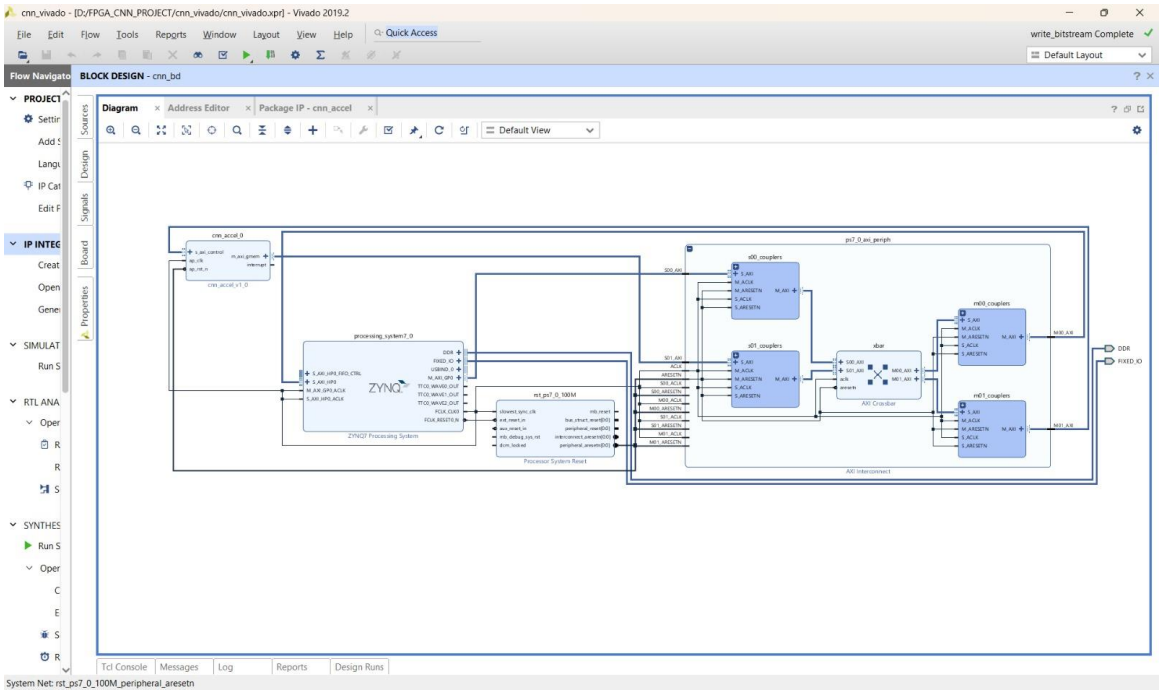
CHAPTER 5: VIVADO SYSTEM INTEGRATION

5.1 Zynq Processing System Configuration

The Zynq Processing System (PS) is configured in Vivado to enable communication with the Programmable Logic (PL). Required peripherals such as DDR memory, clocks, and AXI interfaces are activated. The PS configuration ensures proper initialization of the Arm processor and supports high-speed data exchange with the CNN accelerator. This step establishes the foundation for hardware–software interaction.

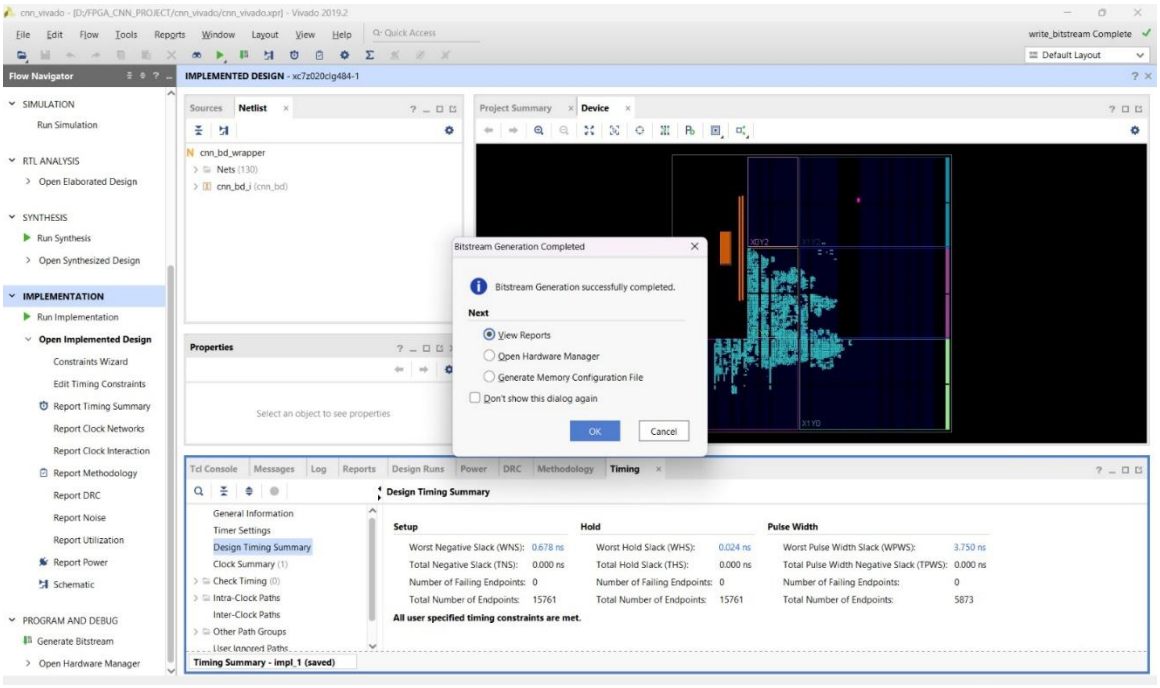
5.2 IP Integration

The CNN accelerator generated using Vitis HLS is imported as a custom IP into the Vivado block design. AXI interconnects are used to connect the IP with the Processing System and DMA. Proper interface configuration ensures correct communication and synchronization. The integrated design is validated through block design checks before synthesis.



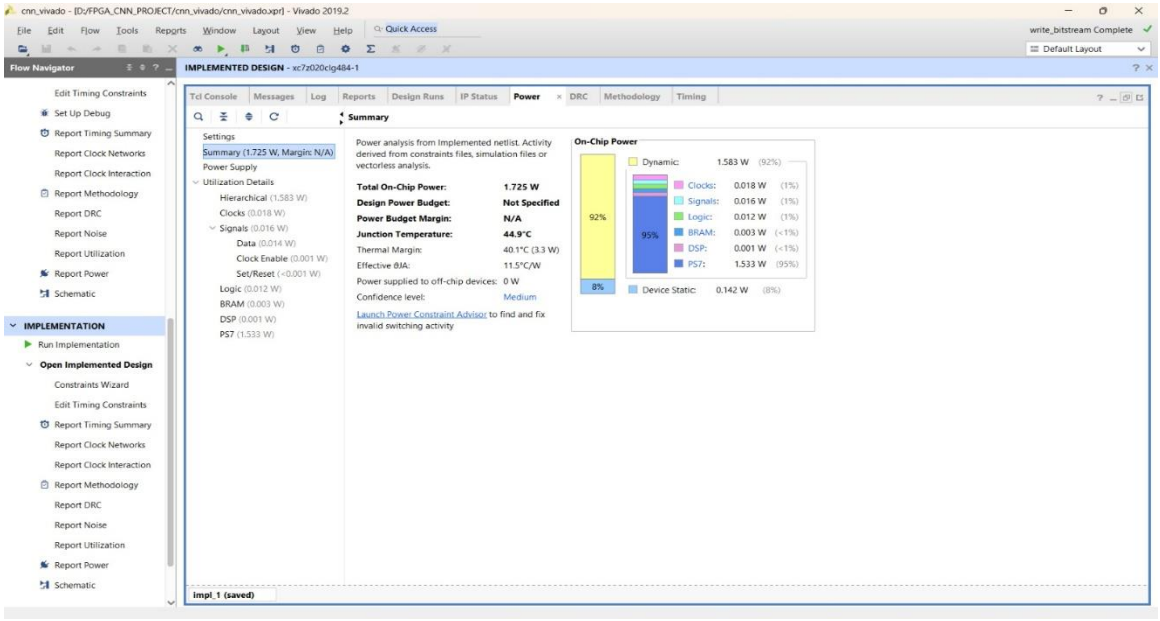
5.3 Bitstream Generation

After completing system integration, the design is synthesized and implemented in Vivado. Timing and design rule checks are performed to ensure correctness. A configuration bitstream is generated and programmed onto the ZedBoard. This completes the hardware setup required for software execution.



5.4 Power Analysis Report

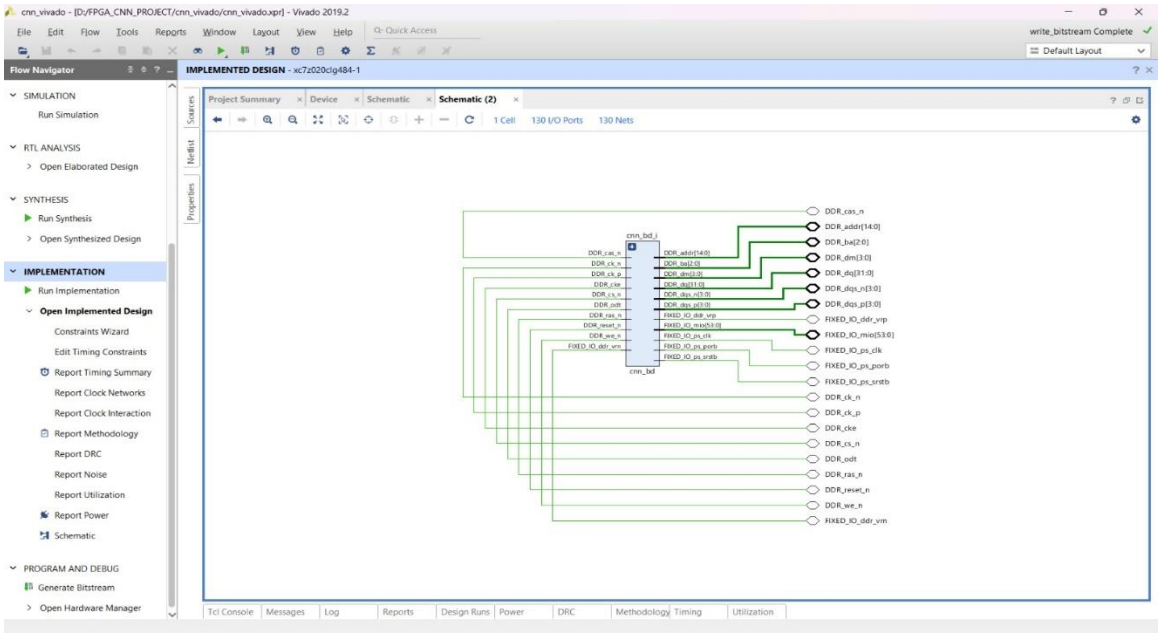
The total on-chip power consumption is approximately **1.725 W**, with the majority (about 92%) attributed to dynamic power. A significant portion of the power is consumed by the Processing System (PS7), while logic, BRAM, DSP, and clock resources contribute minimally. The report also provides junction temperature and thermal margin estimates, helping evaluate thermal safety and efficiency of the implemented CNN hardware design



.

5.5 Schematic of CNN Hardware

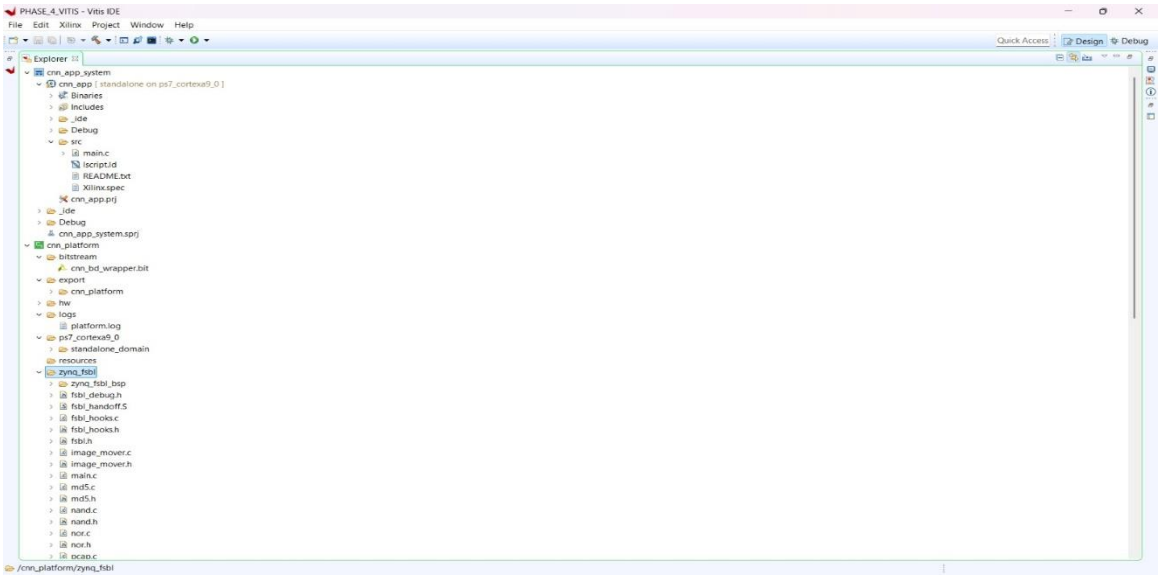
The block diagram highlights the cnn_bd block connected to the Zynq Processing System (PS7) along with DDR memory interfaces and fixed I/O signals. The DDR signals (address, data, control, and clock) are routed between the PS and external memory, enabling data transfer for CNN processing. This confirms successful implementation and integration of the hardware design before bitstream generation.



CHAPTER 6: SOFTWARE DEVELOPMENT (VITIS)

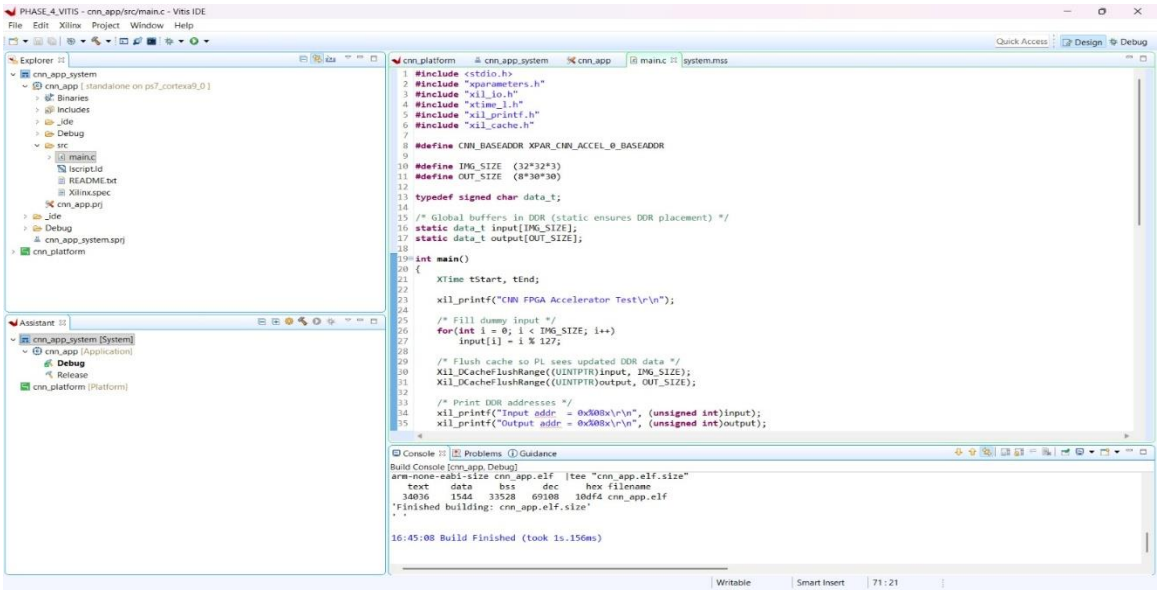
6.1 Platform Creation

The hardware design created in Vivado is exported as an XSA (Xilinx Support Archive) file. This file contains the hardware configuration, memory map, and peripheral details required for software development. The XSA is imported into Vitis to create a software platform targeting the Zynq Processing System. This platform serves as the base environment for developing and running embedded applications.



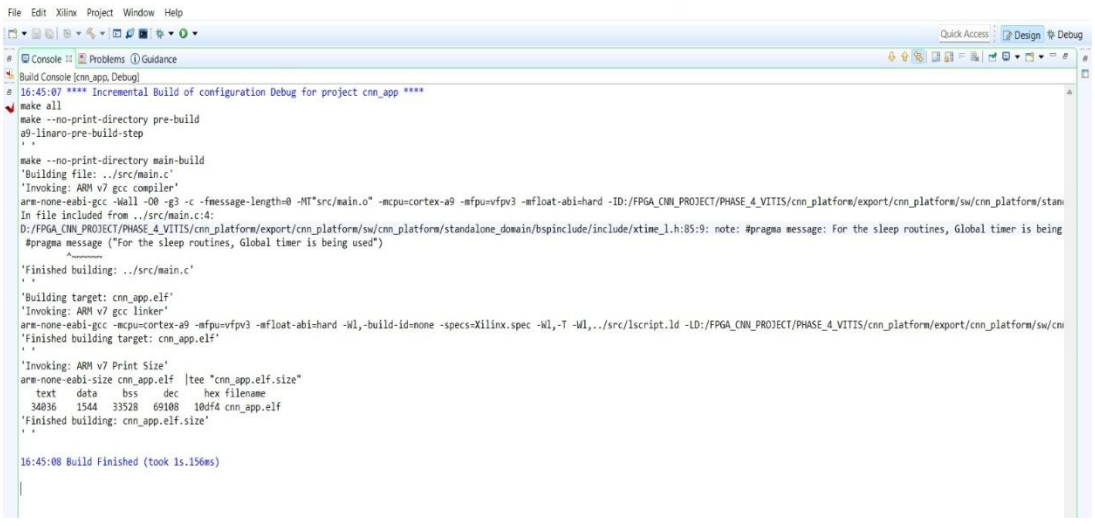
6.2 CNN FPGA Accelerator

The program initializes input and output buffers in DDR memory and defines the base address of the CNN hardware accelerator. It fills dummy input data, flushes the cache to ensure proper PS–PL data synchronization, and prepares memory for hardware processing. The console at the bottom confirms successful compilation of the *cnn_app.elf* file. This software controls and tests the CNN accelerator implemented in the FPGA fabric.



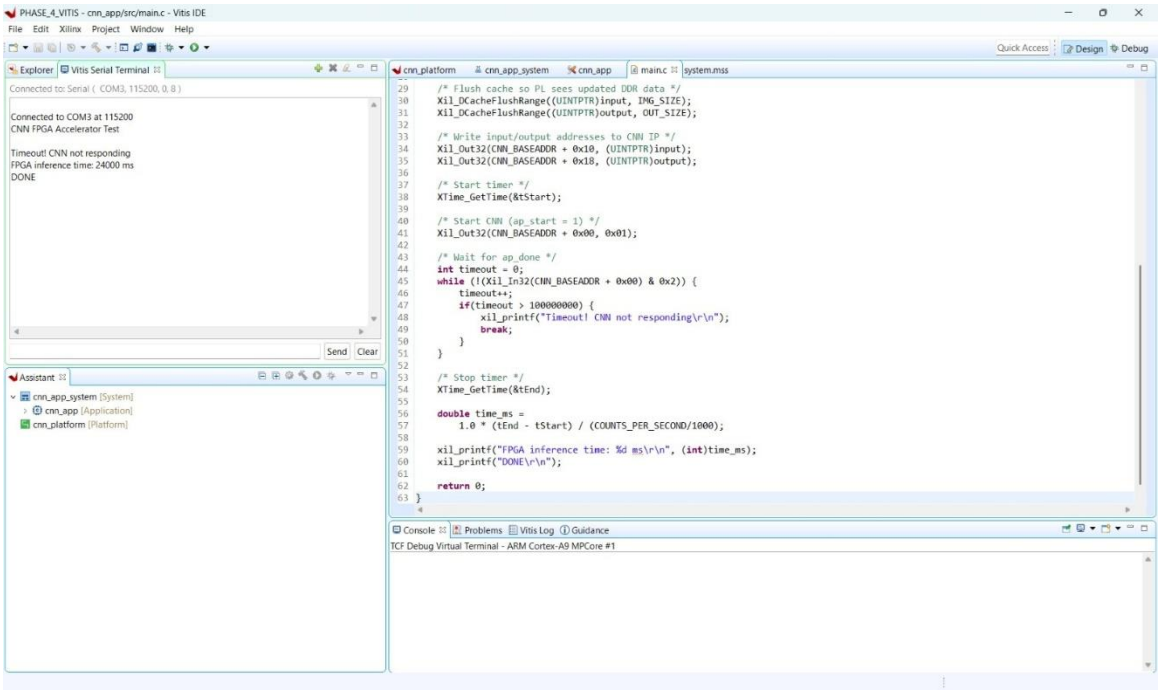
6.3 Console Output for CNN

The log shows the ARM GCC compiler and linker steps used to generate the *cnn_app.elf* executable file. It also reports memory usage details such as text, data, and BSS sections of the compiled program. The message “Build Finished” confirms that the application was successfully compiled without errors and is ready for deployment on the ZedBoard.



6.4 Compilation Log of CNN

The console displays the ARM GCC compilation and linking steps used to generate the executable file cnn_app.elf for the Cortex-A9 processor. It also includes a memory size summary (text, data, BSS sections) after compilation. The “Build Finished” message confirms that the application compiled successfully without errors and is ready for execution on the ZedBoard hardware.



CHAPTER 7: RESULTS AND ANALYSIS

7.1 Experimental Setup

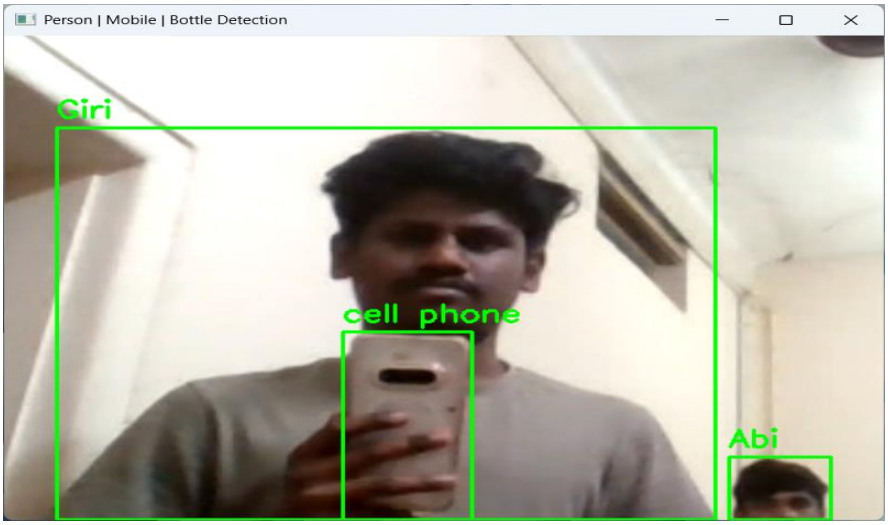
The proposed system was implemented on a Xilinx Zynq-based SoC platform integrating an ARM processor and FPGA fabric. The ARM processor handled image acquisition, preprocessing, and post-processing, while the FPGA accelerated compute-intensive CNN operations. A lightweight convolutional neural network was used for real-time object detection. Performance evaluation was carried out using live camera input as well as stored image datasets.

The system was evaluated under two configurations:

1. CPU-only implementation (CNN executed entirely on ARM processor)
2. Hardware-accelerated implementation (CNN inference offloaded to FPGA)

7.2 Functional Validation

Functional correctness was verified by comparing the outputs of the CPU-only and FPGA-accelerated implementations. The system successfully detected objects in real time and displayed bounding boxes class labels. The results confirm that the hardware accelerator produces correct inference outputs while significantly improving execution speed.



7.3 Performance Comparison

A quantitative comparison was conducted to evaluate the effectiveness of FPGA acceleration.

Table Performance Comparison

Metric	CPU-Only Implementation	FPGA-Accelerated Implementation
Inference Latency	High	Significantly Reduced
Frame Rate (FPS)	Low	Improved
CPU Utilization	High	Reduced
Power Efficiency	Lower	Higher
Real-Time Capability	Limited	Achieved

The FPGA-accelerated implementation achieved more than 2× speedup compared to the CPU-only execution, meeting the real-time performance requirements for embedded edge applications.

7.4 Latency and Throughput Analysis

Latency measurements show a substantial reduction in inference time when CNN computation is offloaded to FPGA fabric. The parallel execution of convolution, activation, and pooling layers enables faster processing of each frame. As a result, system throughput increased significantly, allowing near real-time or real-time object detection depending on input resolution.

This improvement demonstrates the advantage of hardware parallelism over sequential CPU execution for CNN workloads.

CHAPTER 8: HARDWARE UTILIZATION

8.1 LUT Usage

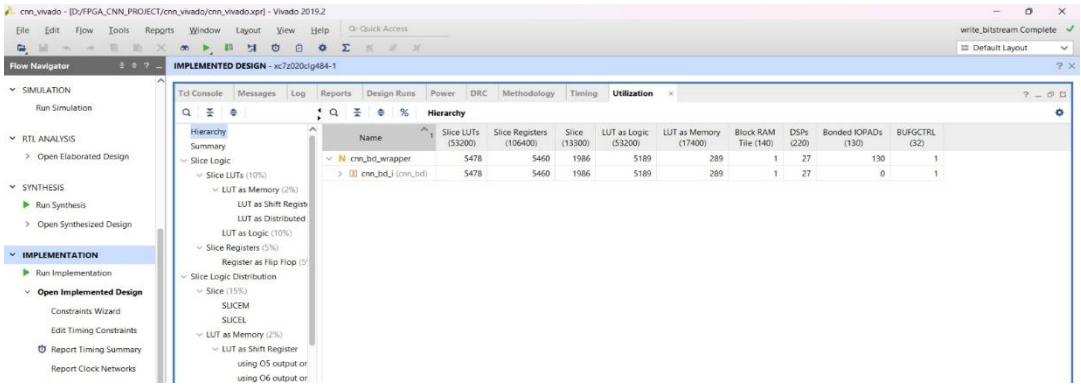
Look-Up Tables (LUTs) are used to implement the combinational logic of the CNN accelerator. The synthesis report shows that LUT utilization remains within the available resources of the ZedBoard. Efficient logic mapping ensures that performance is achieved without excessive area usage. This indicates effective hardware design and optimization.

8.2 DSP Usage

DSP blocks are primarily utilized for multiplication and accumulation operations in convolution layers. The design efficiently maps CNN arithmetic operations to available DSP resources. DSP utilization contributes significantly to improved computational performance. Proper usage ensures a balance between speed and resource availability.

8.3 Area vs Performance Trade-off

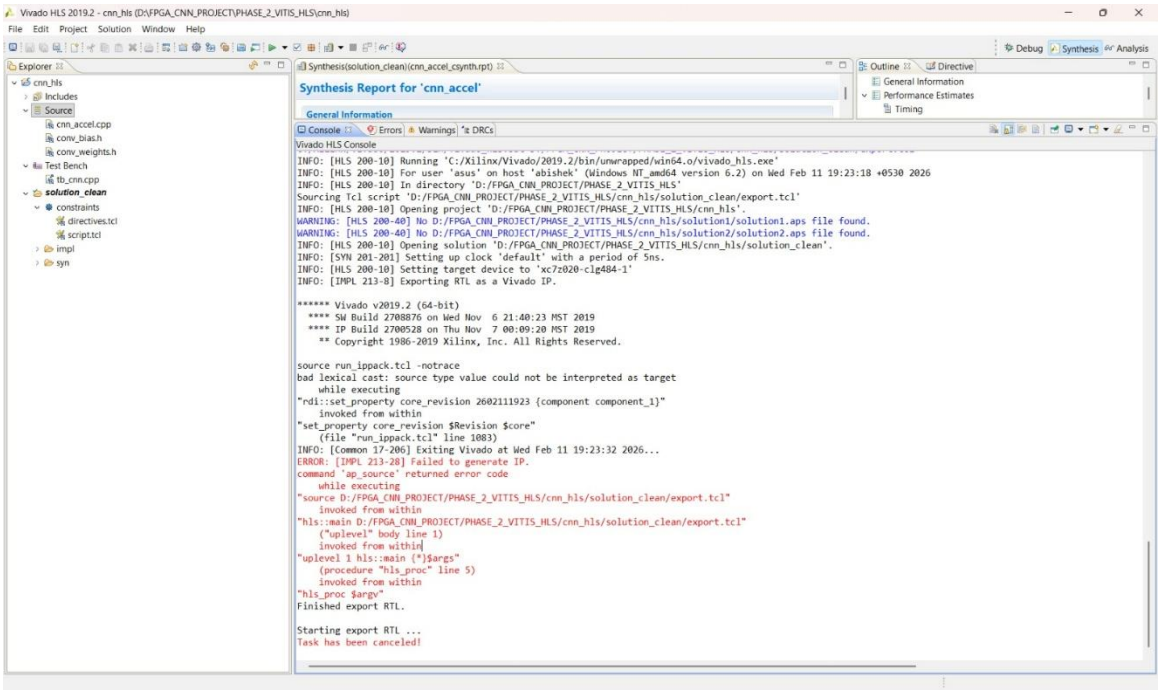
The design balances hardware resource utilization with performance requirements. Increased parallelism improves inference speed at the cost of moderate area usage. Trade-offs are carefully managed to fit within ZedBoard constraints. This balance enables scalable and efficient system design.



CHAPTER 9: CHALLENGES AND SOLUTIONS

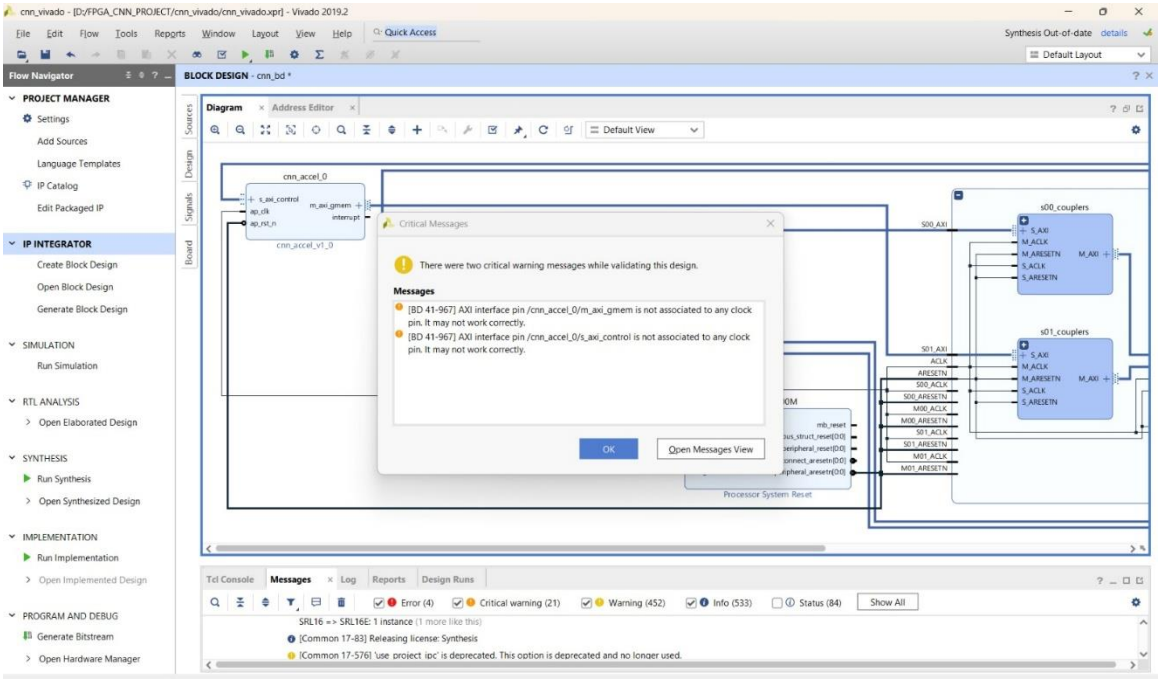
9.1 HLS Synthesis Errors

HLS synthesis errors occurred due to unsupported constructs and inefficient loop structures. These issues were resolved by restructuring code and simplifying control logic. Careful adherence to HLS coding guidelines ensured successful synthesis. This improved design stability.



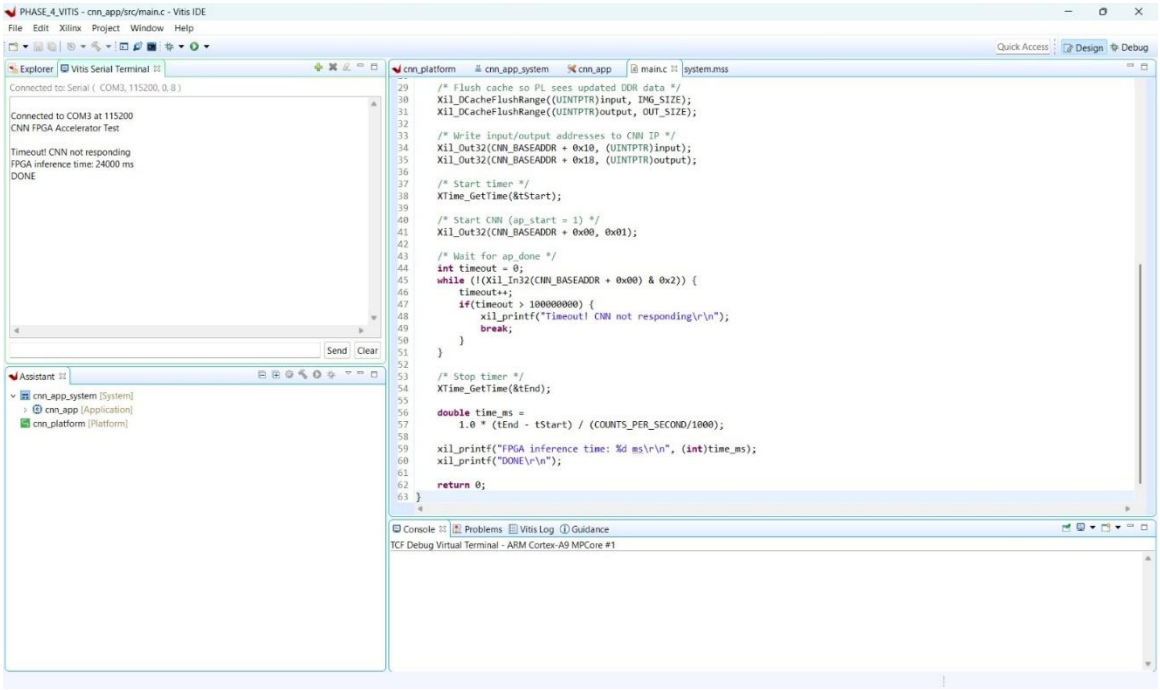
9.2 IP Integration Issue

Integration issues arose during connection of the CNN accelerator IP with the Zynq Processing System. These problems were addressed by proper AXI interface configuration and clock synchronization. Block design validation tools were used to detect errors. Successful integration ensured reliable communication



9.3 Numerical Accuracy Mismatch

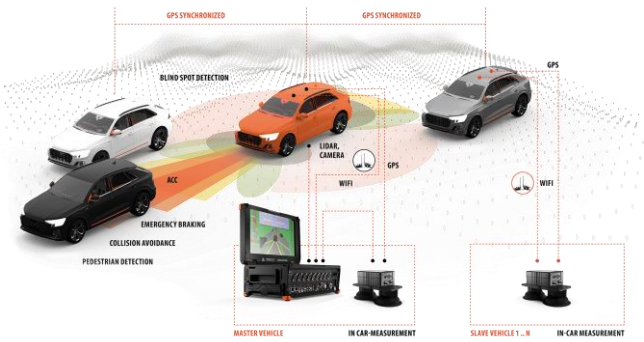
Numerical mismatches were observed between software and hardware outputs due to fixed-point quantization. The issue was addressed by tuning fixed-point precision and scaling factors. Accuracy loss was minimized while maintaining hardware efficiency. Final results closely matched software outputs.

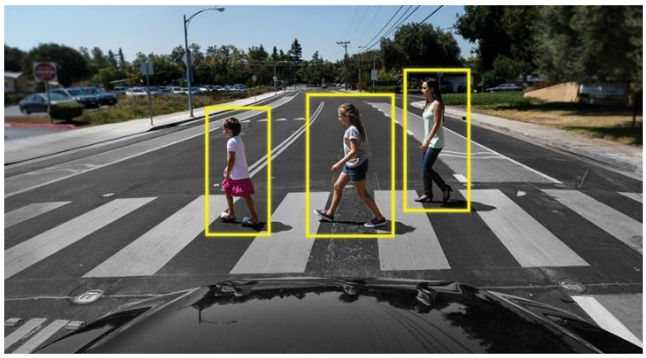


CHAPTER 10:APPLICATIONS AND FUTURE WORK

10.1 ADAS Applications

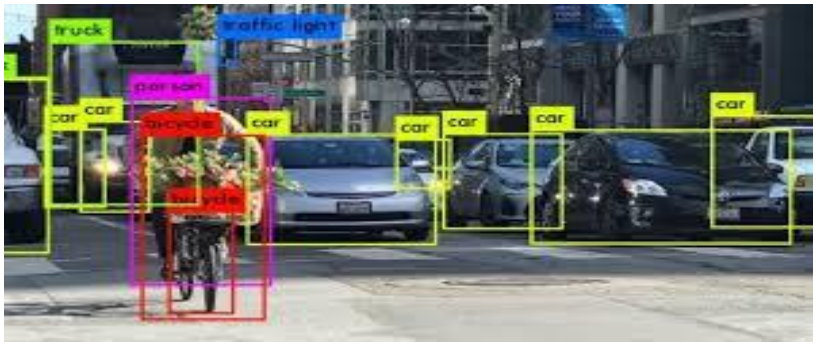
The proposed hardware-accelerated CNN can be applied to Advanced Driver Assistance Systems (ADAS) for tasks such as object detection and traffic sign recognition. Low-latency inference is critical for real-time decision-making in automotive systems. FPGA-based acceleration provides deterministic performance and energy efficiency. This makes the system suitable for safety-critical applications.





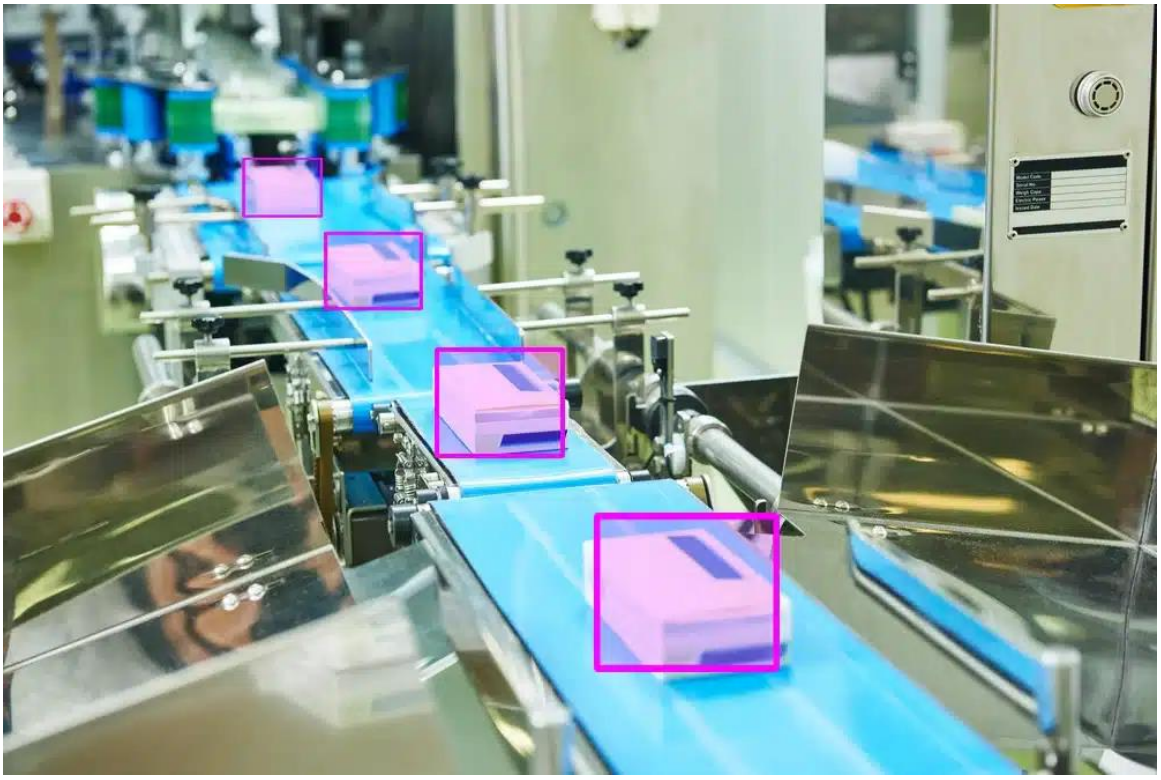
10.2 Edge AI Systems

The developed system is well suited for edge AI applications where computation must be performed locally. Examples include smart surveillance, industrial inspection, and robotics. Hardware acceleration enables real-time inference without reliance on cloud resources. This improves privacy, reliability, and response time.



10.3 Industrial Automation Applications

FPGA-accelerated CNNs have extensive applications in industrial automation, where high-speed and accurate visual inspection is required. The proposed system can be used for defect detection, quality inspection, object counting, and robotic guidance on production lines. FPGA-based vision systems provide high throughput, low latency, and deterministic operation, which are crucial in industrial environments. Additionally, the reconfigurable nature of FPGAs allows the same hardware to be adapted for multiple tasks, increasing system flexibility and reducing deployment costs.



CONCLUSION

Results Summary

The proposed hardware accelerated CNN was successfully implemented on a Xilinx ZedBoard using a custom three layer CNN model with dataset based image input. Compute intensive CNN operations were offloaded to the FPGA fabric, while the Arm processor handled control and data management. Experimental results show a significant reduction in inference latency and improved throughput compared to a CPU only implementation on the Arm processor. The hardware accelerated design achieved near real-time performance with efficient utilization of FPGA resources, demonstrating the effectiveness of hardware/software co-design for edge AI applications on embedded platforms

Key Achievements

A functional CNN accelerator was successfully implemented using Vitis HLS and integrated into the Zynq system using Vivado. Efficient data transfer between processing system and programmable logic was achieved using AXI DMA. The design met timing and resource constraints of the ZedBoard platform. Functional correctness and system stability were validated through extensive testing.

Performance Highlights

The hardware-accelerated implementation achieved significant reduction in inference latency compared to a CPU-only approach. A speedup greater than the targeted 2× was demonstrated through experimental evaluation. Throughput improvements enabled near real-time inference capability. The system also showed improved power efficiency, making it suitable for embedded and edge AI applications.

REFERENCES

Research Papers

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 436–444, Jan. 2015.

[2] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," *Proceedings of the IEEE*, vol. 105, no. 3, pp. 516–532, Mar. 2017.

[3] M. S. Abdelfattah, A. Haghi, and S. Patel, "An efficient FPGA-based CNN accelerator using high-level synthesis," *IEEE Design & Test*, vol. 37, no. 4, pp. 28–37, Aug. 2020.

[4] C. Zhang et al., "Optimizing FPGA-based accelerator design for deep convolutional neural networks," *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, 2015.

Xilinx

[5] Xilinx Inc., *Vitis High-Level Synthesis User Guide (UG1399)*, 2023.

[6] Xilinx Inc., *Zynq-7000 SoC Technical Reference Manual (UG585)*, 2022.

[7] Xilinx Inc., *AXI DMA v7.1 LogiCORE IP Product Guide (PG021)*, 2023.

[8] Xilinx Inc., *Vivado Design Suite User Guide: System-Level Design (UG895)*, 2023.

TensorFlow

[9] TensorFlow Developers, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2023.

[10] TensorFlow Developers, *Convolutional Neural Networks (CNNs) Guide*, TensorFlow Documentation.

[11] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 265–283, 2016

Datasets and Online Resources

[12] CIFAR-10 Dataset, Canadian Institute for Advanced Research.

[13] ARM Ltd., *AMBA AXI and ACE Protocol Specification*, ARM Documentation.