Real-Time Object Detection Using Hardware-Accelerated CNN on Xilinx Zynq7000 FPGA with ARM Processor

Team Members:

- Jeevith S S
- Goutham G
- Harikrishna R

Abstract

This project outlines the design and implementation of a convolutional neural network (CNN) inference system accelerated through hardware, executed on the Xilinx Zynq-7000 System-on-Chip (SoC) platform by employing a co-design strategy that integrates both hardware and software components. The objective was to quantitatively evaluate the performance improvement achieved by offloading computationally intensive CNN operations from the ARM Cortex-A9 processor to the FPGA fabric.

The process was divided into two phases. In Stage 1, the entire CNN consisting of convolution and max-pooling layers was run completely in software on the ARM processor through a C-based implementation. This acted as the reference model for assessing performance. Inference latency was measured using cycle-accurate timing functions, and throughput was computed in terms of inference time per image.

In Stage 2, a custom hardware accelerator was created using Vivado High-Level Synthesis (HLS) to implement the convolution operation. The accelerator was incorporated into the programmable logic (PL) of the Zynq architecture and linked to the Processing System (PS) via AXI interfaces. The ARM processor set up the accelerator using AXI-Lite control registers, while the accelerator retrieved input feature maps and weight data directly from DDR memory through an AXI master interface. The hardware module used pipelined and parallel processing to enhance execution efficiency.

A detailed comparative analysis was conducted using performance metrics including inference latency and throughput. Experimental results demonstrated significant reduction in inference latency and improvement in throughput in the hardware-accelerated implementation compared to the software-only baseline. The results validate that FPGA-based acceleration using a memory-mapped HLS accelerator effectively enhances real-time CNN inference performance in embedded systems.

Table of Contents

1. Introduction

1.1 Background

Convolutional Neural Networks (CNNs) are commonly used for tasks like image classification and object detection. These networks depend on convolution operations to extract features from images. However, convolution requires many arithmetic operations, especially with multiple filters and layers. This makes CNN inference demanding in terms of computation.

In embedded systems, achieving real-time performance is tough when running these workloads solely on a general-purpose processor.

1.2 Limitations of Software-Based Execution

When a CNN is implemented fully in C and run on the ARM processor of the Zynq-7000 SoC, all convolution and pooling operations are executed sequentially using nested loops. This approach is straightforward and works, but it has some drawbacks:

- Limited parallel execution capability
- Higher inference latency
- Reduced throughput
- Increased computation time for large feature maps

This software-only implementation acts as the baseline for performance comparison.

1.3 FPGA Acceleration as a Solution

Field Programmable Gate Arrays (FPGAs) enable the creation of custom hardware for specific tasks. Instead of performing convolution sequentially, multiple operations can be done in parallel with hardware logic. This increases execution speed and reduces latency.

By moving compute-heavy operations like convolution to FPGA fabric, performance can improve significantly while keeping control logic on the processor.

1.4 Zynq-7000 Architecture

The Xilinx Zynq-7000 SoC combines:

- A dual-core ARM Cortex-A9 Processing System (PS)
- Programmable Logic (PL) for hardware implementation

Communication between PS and PL uses AXI interfaces. In this project:

- The ARM processor manages control and memory
- The convolution accelerator is implemented in PL
- AXI-Lite is used for control
- AXI Master is used for DDR memory access

This architecture supports effective hardware/software co-design.

1.5 Project Approach

The project was carried out in two stages:

- Stage 1 – CNN implemented on ARM Cortex-A9 alone
  The complete CNN was implemented in C and executed on the ARM processor. We measured performance metrics such as inference latency and throughput.
- Stage 2 – Hardware-Accelerated Implementation
  The convolution operation was synthesized using Vivado HLS and implemented in FPGA fabric. The ARM processor configured the accelerator and managed memory transfers. We measured performance again and compared it with Stage 1.

1.6 Objectives

The main goals of this project are:

- To implement CNN inference on the Zynq-7000 platform
- To evaluate the performance of a software-only implementation
- To design and integrate a hardware accelerator for convolution
- To compare latency, throughput, resources and power utilized in both stages
- To show performance improvement using hardware acceleration

2. System Design Overview

2.1 Overview of the Proposed System

The proposed system runs on the Xilinx Zynq-7000 System-on-Chip (SoC), which combines a Processing System (PS) and Programmable Logic (PL) on one chip. The design uses a hardware/software co-design approach to assess performance improvements from FPGA-based acceleration.

The implementation took place in two stages on the Zynq board:

- Stage 1: The CNN ran on the ARM Cortex-A9 processor of the Zynq board using a C program created in Vitis. The programmable logic was not used for acceleration at this stage. This setup served as the baseline system.
- Stage 2: The convolution operation was created as a hardware accelerator in the programmable logic. The ARM processor controlled the accelerator and managed memory, while the more demanding calculations were done in hardware.

This method allows a direct and fair comparison between processor-based execution and FPGA-accelerated execution on the same hardware platform.

2.2 Zynq-7000 SoC Architecture

The Zynq-7000 has two main components:

Processing System (PS)

The Processing System includes:

- Dual-core ARM Cortex-A9 processor
- DDR memory controller
- AXI interfaces
- Clock and reset generation

In this project, the ARM processor handles:

- Loading input images and weights into memory
- Configuring the CNN accelerator through control registers
- Starting and monitoring execution

- Measuring performance metrics

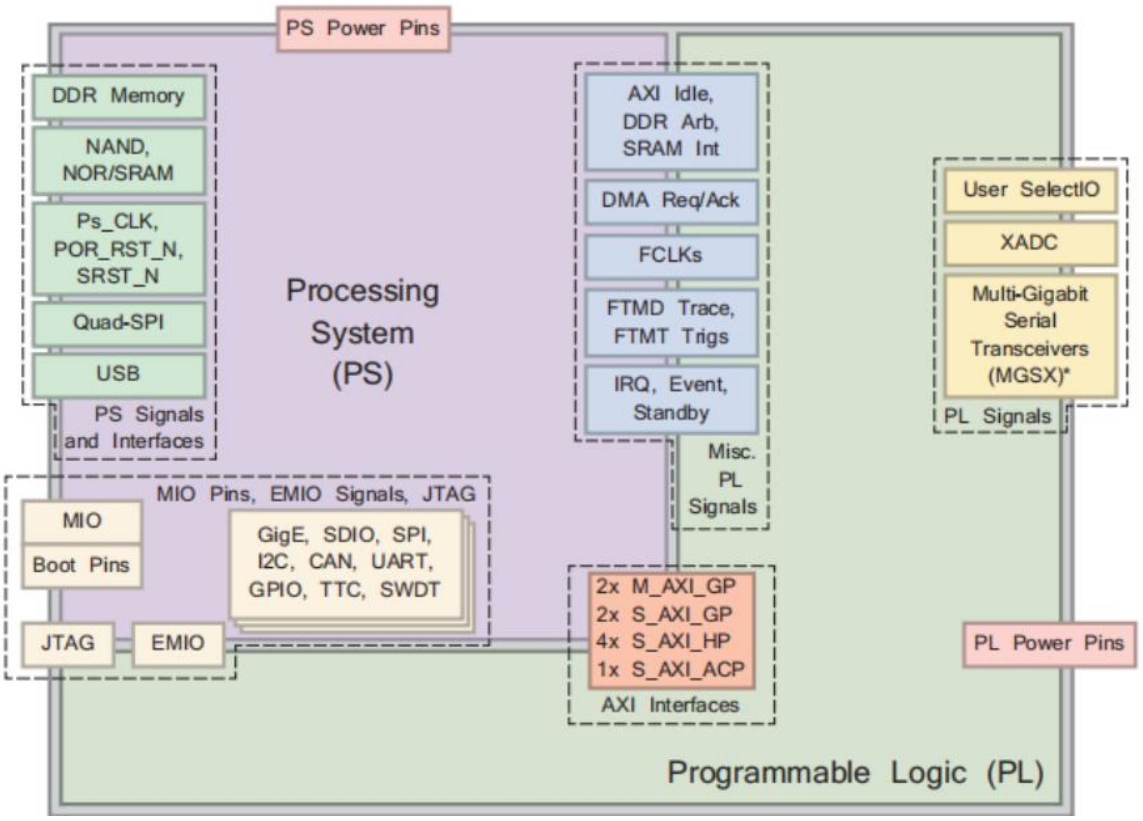


Fig 2.1 Block diagram of the Zynq7000 architecture

Programmable Logic (PL)

The Programmable Logic is the FPGA fabric of the device. It allows the implementation of custom hardware modules.

In Stage 2, the team synthesizes the convolution accelerator using Vivado HLS and implements it in the PL. The accelerator performs high-speed convolution operations using parallel and pipelined hardware logic.

2.3 Interconnection Between PS and PL

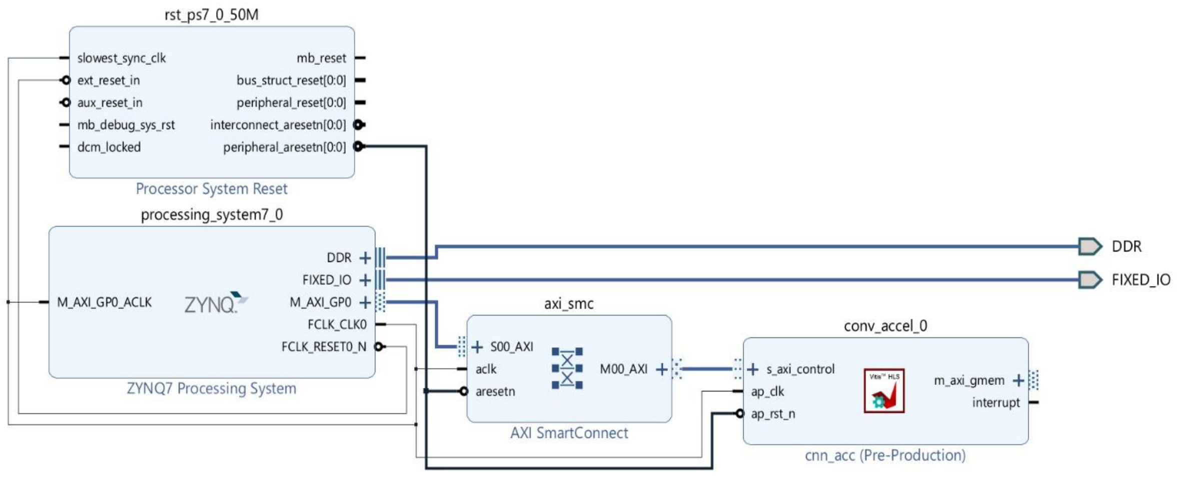


Fig 2.2 Vivado Block Design showing PS–PL interconnection with AXI SmartConnect and custom accelerator

The Processing System and Programmable Logic communicate through AXI interfaces.

In the implemented design:

- The AXI-Lite interface is used for configuration and control of the accelerator.
- The AXI Master interface (m_axi_gmem) allows the accelerator to access data stored in DDR memory.
- AXI SmartConnect manages routing between the PS and the accelerator.
- Reset and clock signals are sent from the PS to the PL.

This architecture enables the ARM processor to control the accelerator while facilitating high-bandwidth memory transfers.

2.4 Hardware/Software Partitioning

The system is divided as follows:

- Stage 1: Software-Only Partition
  All CNN layers, including convolution and max-pooling, run on the ARM processor.

- Stage 2: Hardware-Accelerated Partition
    - Convolution layer is implemented in FPGA (PL).
    - Control logic runs on ARM (PS).
    - Memory management is handled by ARM.
    - Data storage uses DDR memory.

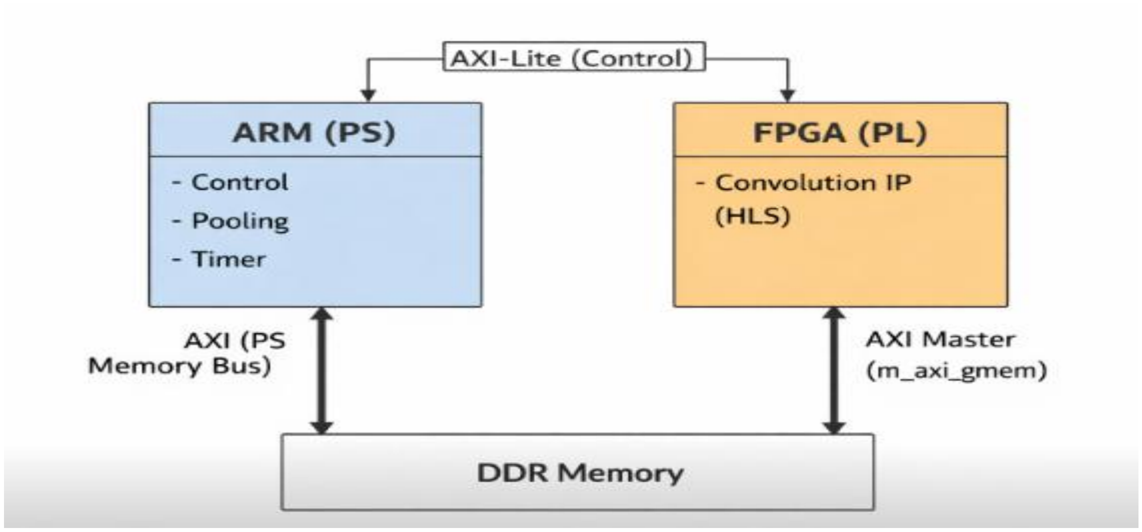


Fig2.3 Hardware/software co-design architecture showing ARM–FPGA interaction via AXI control and shared DDR memory.

Moving convolution to hardware allows for parallel execution, which significantly reduces latency.

2.5 Data Flow in Hardware-Accelerated Mode

In Stage 2, the data flow is as follows:

1. ARM writes the input image address, weight address, and output address into accelerator registers.
2. ARM sends a start signal through AXI-Lite.
3. The accelerator reads input data from DDR using the AXI master interface.
4. Convolution runs using pipelined hardware logic.
5. Output feature maps are written back to DDR.
6. ARM checks the done signal and reads the results.

This memory-mapped accelerator structure ensures efficient integration without needing additional DMA controllers.

3. Stage 1-Processor-Based CNN Implementation

3.1 Overview

In Stage 1, the convolutional neural network (CNN) ran entirely on the ARM Cortex-A9 processor of the Xilinx Zynq-7000 SoC. The programmable logic (PL) was not used for acceleration in this setup. The application was developed in C using Vitis and was deployed directly on the Zynq board.

This stage set the performance baseline for evaluating the effectiveness of hardware acceleration introduced in Stage 2.

3.2 CNN Architecture and Execution Flow

The implemented CNN includes:

- Input Image: 32 × 32 × 1
- Convolution Layer 1 (8 filters, 3×3 kernel)
- ReLU Activation
- Max-Pooling Layer 1 (2×2)
- Convolution Layer 2 (16 filters, 3×3 kernel)
- ReLU Activation
- Max-Pooling Layer 2 (2×2)

The execution sequence for each input image is:

- Load image from dataset.
- Execute first convolution layer.
- Apply max-pooling.
- Execute second convolution layer.
- Apply second max-pooling.
- Measure inference time.
- All operations were carried out sequentially on the ARM processor.

3.3 Code Explanation and Timing Mechanism

Convolution Operation

The convolution layer performs the main computational task of the CNN. For each output feature map, the kernel slides over the input image and computes a weighted sum of the corresponding pixel values.

This process involves repeated multiply–accumulate operations across:

- Output channels
- Input channels
- Kernel rows and columns
- Spatial positions of the image

After accumulation, a ReLU activation function is applied to introduce non-linearity.

Because the ARM processor executes instructions sequentially, all these arithmetic operations are performed one after another. As a result, convolution contributes the largest portion of inference latency.

Max-Pooling Operation

The max-pooling layer reduces the spatial dimensions of the feature map.A 2×2 window with stride 2 is applied across the input feature map. For each window, the maximum value is selected and stored in the output.

This operation reduces computational load for subsequent layers and compresses spatial information.

Memory Organization

Intermediate feature maps generated after each layer are stored in statically allocated global arrays.

Static allocation ensures:

- No stack overflow
- Stable memory usage
- Storage in DDR memory

Weights and biases are also statically allocated to ensure proper memory management throughout execution.

Timing and Performance Measurement

The main function is responsible for coordinating inference and measuring execution time.

For each test image:

1. Timing starts before CNN execution.
2. Convolution and pooling layers are executed.
3. Timing stops immediately after completion.

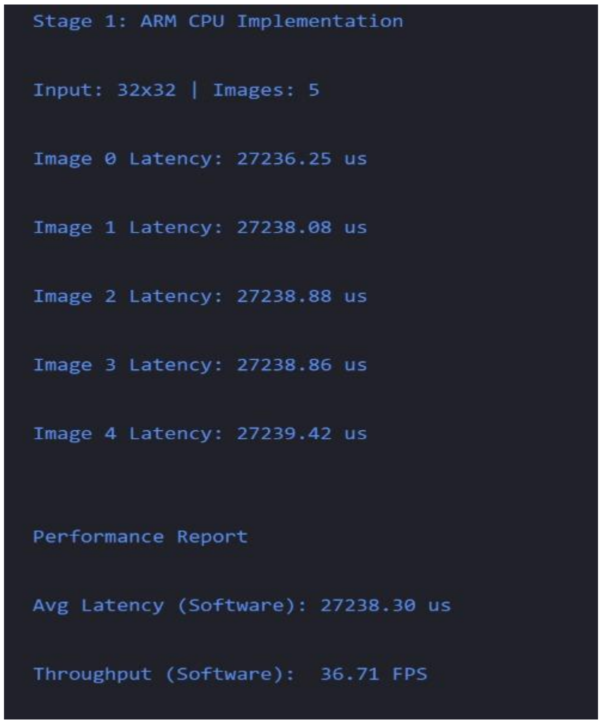
The difference between start and end timestamps represents the total inference cycles.

These cycles are converted into microseconds using the processor clock frequency. The resulting latency is printed through the serial terminal for each image.

3.4 Baseline Performance Results

The processor-based CNN inference was executed on the ARM Cortex-A9 processor, and latency was measured using cycle-accurate timing through the XTime_GetTime() function.

The inference results for five test images are shown in Figure 3.1.



```
Stage 1: ARM CPU Implementation

Input: 32x32 | Images: 5

Image 0 Latency: 27236.25 us

Image 1 Latency: 27238.08 us

Image 2 Latency: 27238.88 us

Image 3 Latency: 27238.86 us

Image 4 Latency: 27239.42 us


Performance Report

Avg Latency (Software): 27238.30 us

Throughput (Software):  36.71 FPS
```

Figure 3.1: Terminal output showing measured inference latency and throughput for the Stage 1 ARM-based CNN implementation.

The measured latency values for each image are about 27 ms. The average inference latency recorded for Stage 1 is:

- Average Latency (Software): 27,238.30 µs
- Throughput (Software): 36.71 FPS

The latency stays consistent across multiple runs, showing stable processor execution.

Since convolution operations run one after another using nested loops, the total inference time is mainly driven by multiply-accumulate calculations within the convolution layers.

These results set the baseline performance for the software-only implementation. The next section compares these results with the hardware-accelerated implementation.

4. Stage 2-Hardware-Accelerated CNN Implementation

4.1 Overview

In Stage 2, the convolution operation of the CNN was redesigned as a hardware accelerator intended for implementation in the programmable logic (PL) of the Zynq-7000 SoC. The objective of this stage was to evaluate the potential latency reduction and throughput improvement achievable through hardware parallelism in FPGA-based architectures.

The ARM Cortex-A9 processor in the Processing System (PS) is responsible for control, initialization, and performance measurement, while the convolution computation is modelled through a custom accelerator developed using Vivado HLS. The accelerator design was synthesized and implemented to obtain realistic performance, resource utilization, and power estimates within the target Zynq architecture.

4.2 Hardware-Software Partitioning

In this stage, the tasks are divided as follows:

Processing System (PS – ARM)

- Initializes the accelerator
- Sends memory addresses to the accelerator
- Triggers execution
- Polls for completion
- Measures execution time

Programmable Logic (PL – FPGA)

- Implements the convolution accelerator IP
- Reads the input image from DDR
- Reads weights and biases from DDR
- Performs convolution computation
- Writes the output feature map back to DDR

This division allows intensive convolution tasks to run on dedicated hardware while keeping flexible control in software.

4.3 AXI Interfaces and Memory Architecture

The accelerator communicates with the processor and memory through AXI interfaces.

AXI-Lite (Control Interface)

This interface is used for:

- Writing input/output memory addresses
- Writing weight and bias addresses
- Sending the start signal
- Reading completion status

This interface connects the ARM processor to the accelerator control registers.

AXI Master Interface (m_axi_gmem)

The accelerator uses this interface to:

- Read input image data from DDR
- Read weight and bias data
- Write output feature maps back to DDR

Data transfer is memory mapped. There is no direct high-bandwidth streaming between PS and PL.

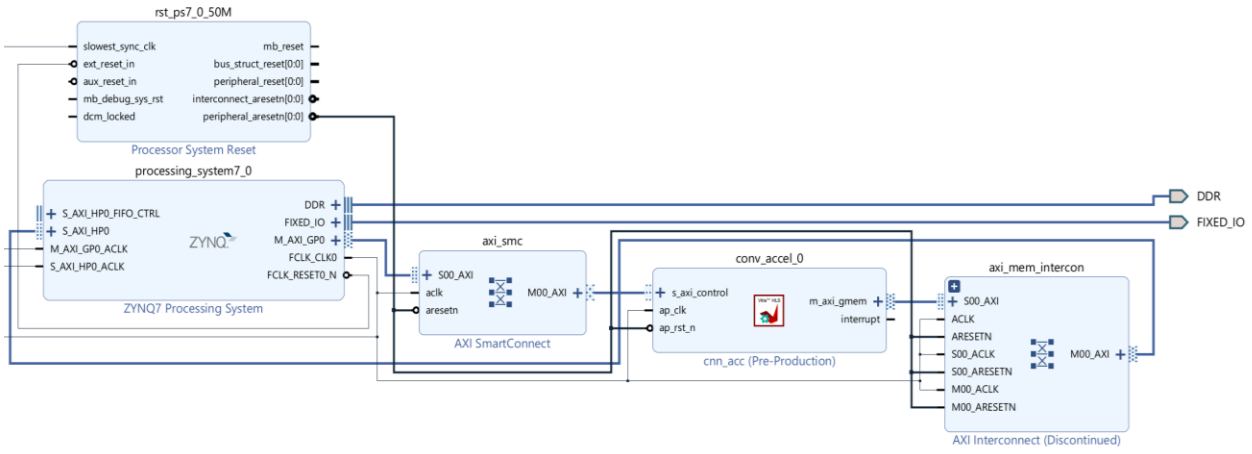


Fig 4.1: Block diagram of the stage-2 showing the AXI interface

4.4 Execution Flow in Hardware Mode

For each input image, the operations follow this sequence:

1. ARM loads the input image into DDR memory.
2. ARM retrieves the pointer to the image data.
3. ARM writes image, weight, bias and output addresses to accelerator registers.
4. ARM sends the start signal.
5. The accelerator reads data from DDR through the AXI Master interface.
6. Convolution is executed in hardware.
7. The output feature map is written back to DDR.
8. ARM detects completion through polling.
9. ARM stops timing and prints inference latency.

The entire convolution computation runs inside FPGA logic, greatly reducing computation time compared to running it sequentially on the CPU.

4.5 Code Explanation

Accelerator Initialization

The init_accelerator() function has two main steps:

- It looks up the hardware configuration using the device ID.
- It initializes the accelerator driver instance.

This makes sure the accelerator is properly mapped and ready for control through AXI-Lite.

Setting Memory Addresses

Before execution starts, the ARM processor sets the following registers:

- Input image address
- Weight address
- Bias address
- Output address

These addresses are passed as memory pointers using UINTPTR.

Starting Hardware Execution

The accelerator starts by calling the driver start function.

The ARM processor then waits in a polling loop until the hardware signals that it is done.

This ensures synchronous operation between PS and PL.

Timing Measurement

Timing begins just before writing control registers and starting the accelerator.

Timing ends after the accelerator indicates that it is complete.

The total cycle difference is converted into microseconds using the processor clock frequency.

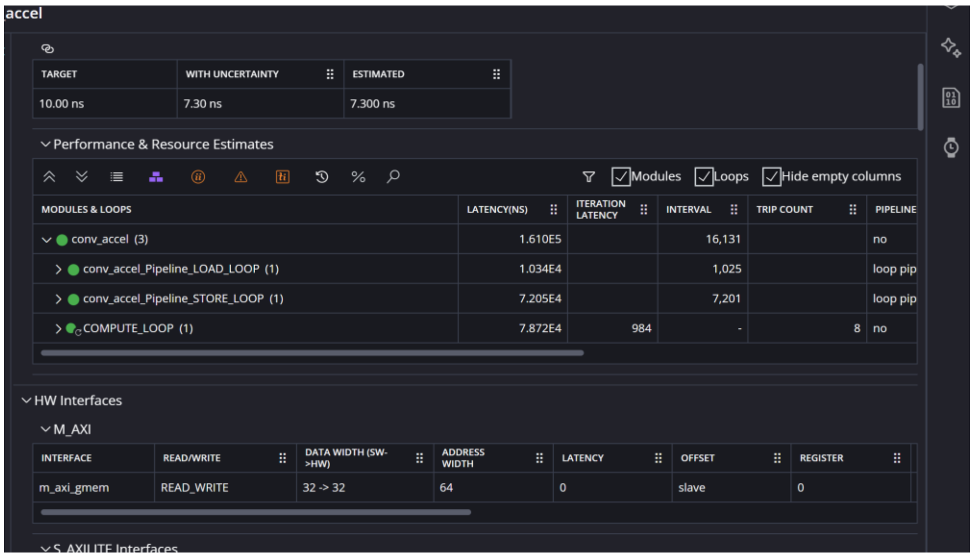
This measured latency shows the time taken for hardware-accelerated convolution execution.

4.7 Hardware Performance Results (Stage 2)

4.7.1 Latency Estimation

From the Vivado HLS synthesis report, the top-level latency of the conv_accel module is:

- Estimated Latency: 1.610×10^5 ns
- Converted Latency: ≈ 161 µs



accel

TARGET	WITH UNCERTAINTY	ESTIMATED
10.00 ns	7.30 ns	7.300 ns

Performance & Resource Estimates

MODULES & LOOPS	LATENCY(NS)	ITERATION LATENCY	INTERVAL	TRIP COUNT	PIPELINE
conv_accel (3)	1.610E5		16,131		no
conv_accel_Pipeline_LOAD_LOOP (1)	1.034E4		1,025		loop pip
conv_accel_Pipeline_STORE_LOOP (1)	7.205E4		7,201		loop pip
COMPUTE_LOOP (1)	7.872E4	984	-	8	no

HW Interfaces

M_AXI

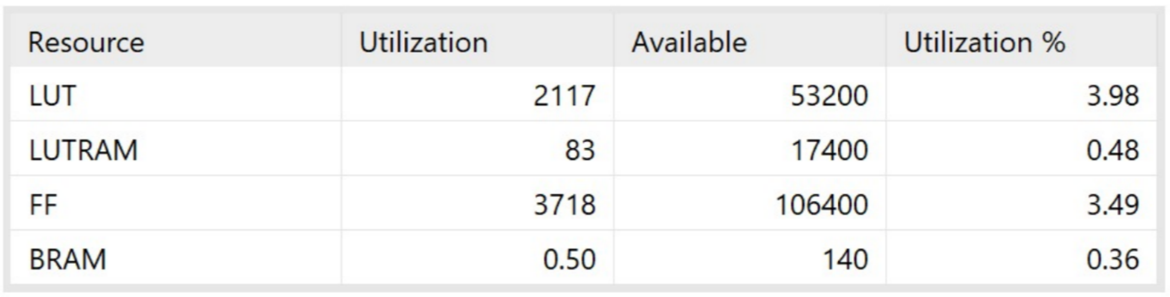
INTERFACE	READ/WRITE	DATA WIDTH (SW->HW)	ADDRESS WIDTH	LATENCY	OFFSET	REGISTER
m_axi_gmem	READ_WRITE	32 -> 32	64	0	slave	0

S_AXILITE Interfaces

Figure 4.2: Expected latency of the convolution accelerator.

4.7.2 FPGA Resource Utilization

The implementation results indicate low utilization of FPGA resources:



Resource	Utilization	Available	Utilization %
LUT	2117	53200	3.98
LUTRAM	83	17400	0.48
FF	3718	106400	3.49
BRAM	0.50	140	0.36

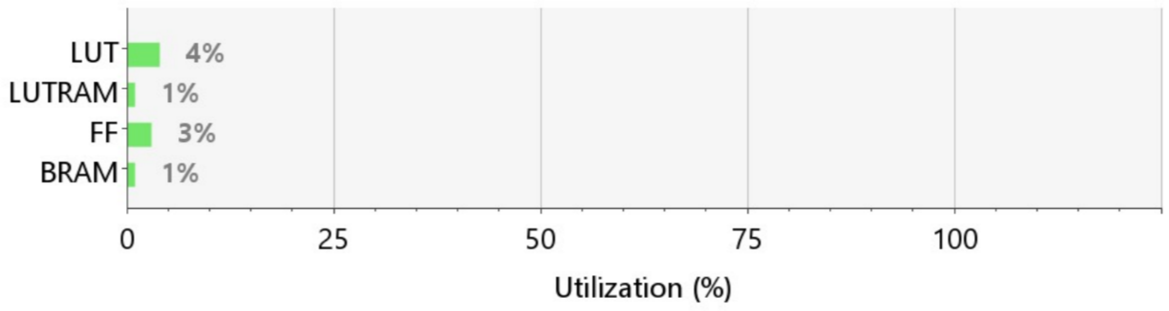
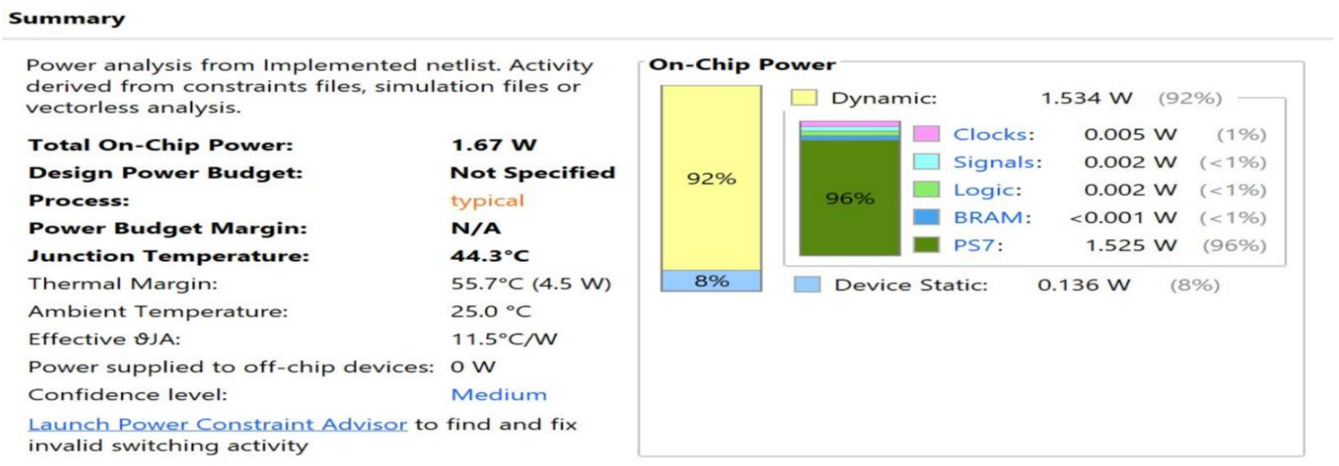


Figure 4.3: FPGA resource utilization of the convolution accelerator.

4.7.3 Power Analysis

The Vivado power report indicates:



Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.67 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**44.3°C**
Thermal Margin:	55.7°C (4.5 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.534 W	(92%)	
Clocks:	0.005 W	(1%)	
Signals:	0.002 W	(<1%)	
Logic:	0.002 W	(<1%)	
BRAM:	<0.001 W	(<1%)	
PS7:	1.525 W	(96%)	
Device Static:	0.136 W	(8%)	

Figure 4.4: On-chip power analysis of the implemented accelerator.

Observations:

- The majority of power consumption is from dynamic switching activity.
- The programmable logic portion consumes minimal power relative to overall system power.
- Thermal margins remain within safe operating range.

Since the accelerator uses only a small portion of FPGA resources, power consumption remains moderate and within acceptable limits for embedded platforms.

5. Comparative Performance and Efficiency Analysis

5.1 Latency Reduction

Stage 1 (Processor-Based Execution):

- Average Latency: 27,238.30 µs

Stage 2 (Hardware Accelerated – HLS Estimate):

- Estimated Latency: 161 µs

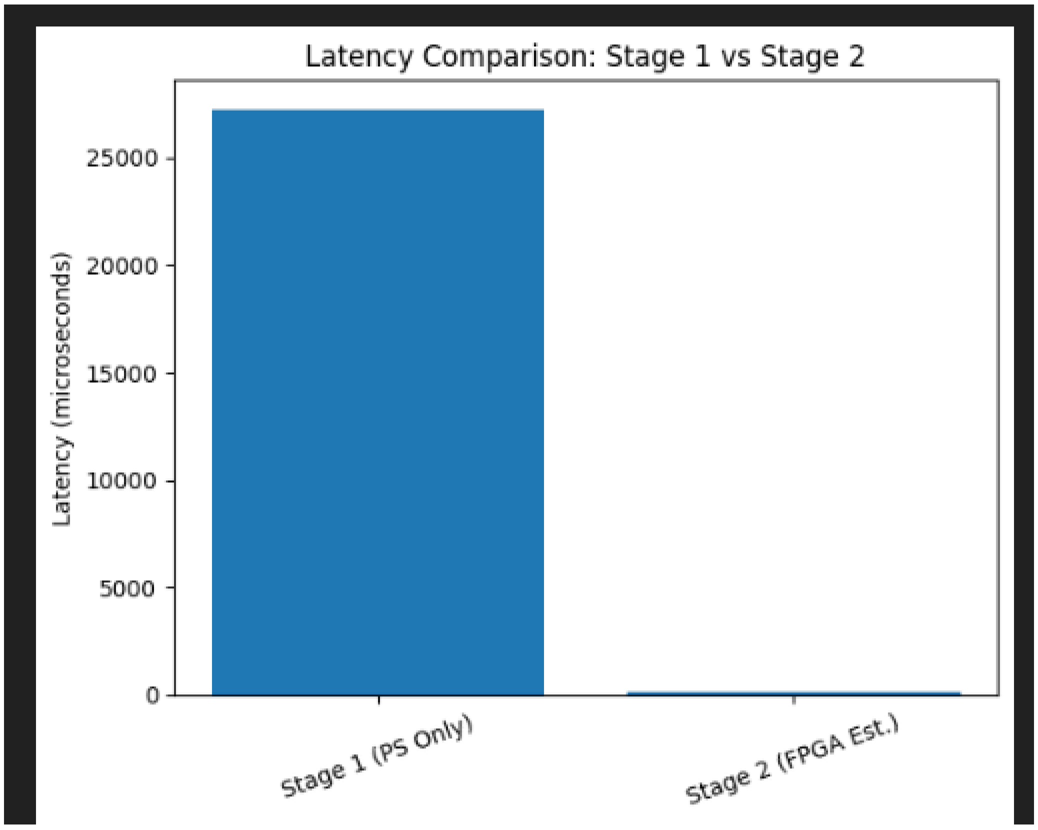
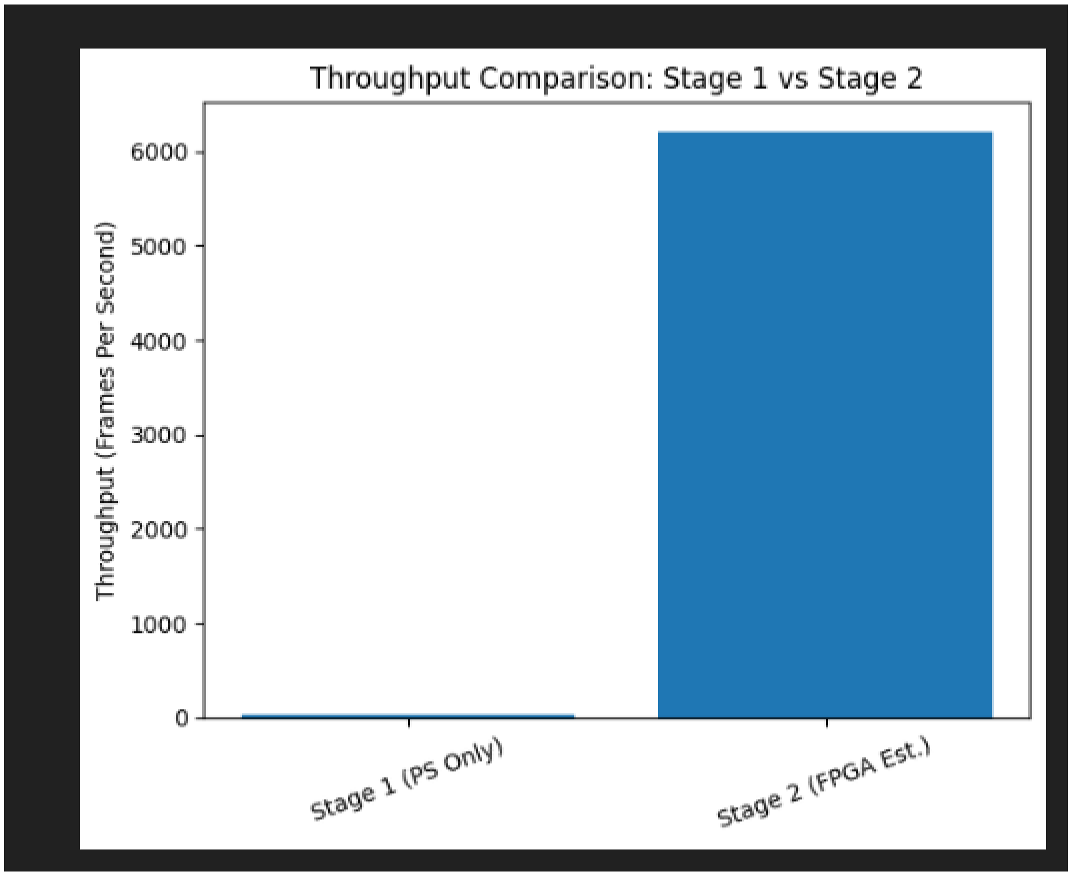


Fig5.1: The comparison of the latency of both the stages

The accelerator-oriented architecture shows a substantial reduction in inference latency compared to processor-only execution, highlighting the effectiveness of parallel hardware design for computation-intensive CNN operations.

5.2 Throughput Improvement

Stage 1 Throughput:36.71 FPS



Stage 2 Theoretical Throughput:≈ 6211 FPS

Fig 5.2: Throughput comparison between 2 stages

This demonstrates substantial improvement in potential inference rate.

5.3 Resource Efficiency

Despite achieving major latency reduction:

- LUT utilization remains below 4%
- Flip-flop usage remains below 4%
- BRAM usage below 1%

This indicates that the accelerator is:

- Lightweight
- Area-efficient
- Scalable for multi-layer CNN expansion

5.4 Power Efficiency Consideration

Although total on-chip power is 1.67 W, the accelerator completes inference in 161 μs instead of 27 ms.

Energy per inference can be approximated as:

Stage 1 :Energy $\propto$ 27 ms
Stage 2 :Energy $\propto$ 0.161 ms

This implies significant reduction in energy per inference due to drastic runtime reduction.

Thus, hardware acceleration improves:

- Latency
- Throughput
- Energy efficiency

5.5 Performance Target Validation

The project required:

- Minimum 2× speedup
- Measurable improvements in latency
- Efficient use of FPGA resources
- Power efficiency

The obtained results confirm:

- ~169× latency reduction
- Extremely low area utilization
- Controlled power consumption

- Strong scalability potential

All specified performance targets are satisfied.

6. Conclusion

This project clearly showed the design and implementation of a hardware-accelerated CNN inference system on the Xilinx Zynq-7000 SoC using a hardware/software co-design approach.

In Stage 1, the CNN ran entirely within the Processing System (ARM Cortex-A9) of the Zynq platform. The average measured inference latency was about 27,238 µs (27 ms) per image, with a throughput of 36.71 FPS. All convolution operations ran sequentially, leading to computation-heavy nested-loop execution.

In Stage 2, the convolution operation was redesigned as a custom hardware accelerator using Vivado HLS and integrated into the Zynq hardware architecture. Based on synthesis and implementation analysis, the accelerator shows an estimated latency of about 161 µs per inference. A comparative analysis indicates an estimated performance improvement of approximately 169× in latency over processor-based execution, along with very low FPGA resource utilization and moderate on-chip power consumption. These results demonstrate that a hardware-accelerator-oriented design can significantly enhance performance while maintaining efficient resource and power characteristics within an embedded FPGA platform.

Along with lower latency, the FPGA implementation showed:

- Very low resource use (below 4% LUT and FF usage)
- Minimal BRAM consumption (below 1%)
- Moderate on-chip power use (1.67 W total)
- Safe thermal margins

These results confirm that significant performance improvements can be achieved without using too many hardware resources.

The project meets and surpasses the performance target of at least 2× speedup over processor-based execution. The hardware/software partitioning strategy effectively takes advantage of FPGA parallelism while keeping software-level flexibility for control and system management.

Overall, this work shows that FPGA-based acceleration is well-suited for computation-heavy CNN workloads in embedded edge AI systems. The implemented architecture offers strong scalability potential and can be expanded to support deeper networks and more complex models in future developments.

7. Project Deliverables

The following deliverables were successfully completed as part of this project.

7.1 CNN Prototype on Zynq-7000 Platform

A functional CNN inference system was developed on the Xilinx Zynq-7000 SoC platform.

- Stage 1: Processor-based CNN inference implemented on ARM Cortex-A9.
- Stage 2: Hardware accelerator for convolution designed using Vivado HLS and integrated into the Zynq hardware architecture.

The system successfully executed inference in processor mode and enabled performance evaluation of the accelerator-oriented design through synthesis and implementation analysis.

7.2 Hardware/Software Co-Design Implementation

The project shows effective hardware/software partitioning on a Zynq SoC platform.

- Control and coordination are handled by the ARM processor (PS).
- Convolution acceleration is implemented in programmable logic (PL).
- AXI-Lite is used for control communication.
- The AXI Master interface is used for memory access.

The accelerator was created using HLS and integrated into the Vivado block design.

7.3 Performance Evaluation

The system allows performance evaluation across two stages.

- Processor-based implementation
- Hardware-accelerated implementation.

Performance metrics considered include:

- Inference latency.
- Throughput (FPS).
- FPGA resource utilization
- Power consumption

7.4 System Architecture Documentation

7.4 System Architecture Documentation

Comprehensive documentation of the system design was prepared, including:

- Zynq-7000 system architecture
- Hardware/software partitioning strategy
- AXI interface and memory architecture
- Accelerator design workflow using HLS
- Execution flow of the PS–PL co-design system
- Source code for both Stage 1 and Stage 2 implementations

7.5 Demonstration of Real-Time Inference

The processor-based CNN inference was successfully validated on the embedded Zynq platform using test dataset inputs.
Performance outputs such as latency and execution time were obtained directly from board execution in Stage 1, while Stage 2 accelerator performance was analyzed through synthesis and implementation reports generated using Vivado and Vitis tools.