BHARAT AI-SoC STUDENT CHALLENGE A Project-based Virtual Challenge to Ignite Innovation in AI-driven System-on-Chip (SoC) Design

Team Members:

1) Balaji K
   212223060028
   BE ECE
   3rd Year
2) Madhavan R
   212223060142
   BE ECE
   3rd Year
3) Om Sharma M
   212223060190
   BE ECE
   3rd Year

Real-Time Object Detection Using Hardware-Accelerated CNN on Xilinx Zynq SoC

Abstract

Edge AI applications require real-time inference under strict power and latency constraints. General-purpose processors often fail to meet these requirements due to limited parallelism and energy inefficiency. This project presents the design and implementation of a hardware-accelerated Convolutional Neural Network (CNN) inference system on a Xilinx Zynq System-on-Chip (SoC), leveraging FPGA fabric for compute-intensive operations and an Arm processor for system control.

A lightweight CNN was implemented using Vitis HLS and integrated as a custom IP into the Zynq programmable logic. The accelerator was deployed on a PYNQ-Z2 board and evaluated against a CPU-only baseline. Experimental results demonstrate more than a 2× speedup in inference latency, increased throughput, and improved power efficiency, validating the effectiveness of hardware/software co-design for edge AI workloads.

1. Introduction

The rapid growth of edge computing has driven demand for efficient on-device artificial intelligence. Applications such as object detection, surveillance, and autonomous systems require low latency and high throughput while operating within limited power budgets. Traditional CPU-based inference struggles to satisfy these constraints, especially for convolution-heavy neural networks.

Field Programmable Gate Arrays (FPGAs) provide massive parallelism, reconfigurability, and energy efficiency, making them ideal for accelerating CNN workloads. Xilinx Zynq SoCs integrate Arm processors with FPGA fabric on a single chip, enabling heterogeneous computing through hardware/software co-design.

This project focuses on implementing a real-time CNN inference pipeline on a Zynq platform, where computationally intensive CNN layers are offloaded to FPGA hardware while the Arm processor manages preprocessing, control, and post-processing.

2. Objectives

The primary objectives of this project are:

- To design and implement a CNN accelerator on FPGA using HLS
- To integrate the accelerator with an Arm-based embedded system
- To achieve real-time or near real-time inference
- To quantitatively compare CPU-only and FPGA-accelerated implementations
- To analyze performance, resource utilization, and power efficiency

3. Related Work

Several studies have demonstrated CNN acceleration using GPUs and specialized ASICs. However, GPUs are power-hungry and unsuitable for embedded edge devices. ASICs offer high efficiency but lack flexibility.

FPGA-based CNN accelerators provide a balance between performance, flexibility, and power efficiency. High-Level Synthesis (HLS) tools such as Vitis HLS enable rapid development of hardware accelerators by abstracting low-level RTL complexity. This project builds upon these principles by implementing a lightweight CNN accelerator optimized for embedded deployment.

4. System Architecture

4.1 Hardware Platform

- Xilinx Zynq-7000 SoC (PYNQ-Z2 Board)
- Dual-core ARM Cortex-A9 Processor

- Programmable Logic with DSP blocks, BRAM, and LUTs
- USB camera / image dataset input

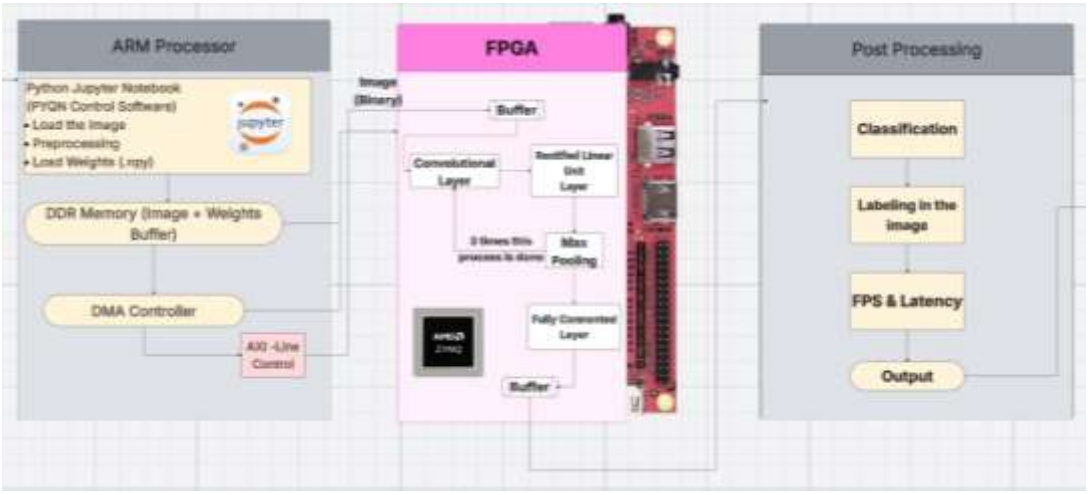
4.2 Software Stack

- Vitis HLS for CNN accelerator development
- Vivado Design Suite for system integration and bitstream generation
- PYNQ Linux OS
- Python and OpenCV for application control and preprocessing

4.3 Overall Architecture

The system follows a heterogeneous computing model:

- The Arm processor handles image acquisition, preprocessing, control logic, and post- processing.
- The FPGA fabric accelerates CNN layers such as convolution, activation (ReLU), and pooling.
- Data transfer between CPU and FPGA is achieved using AXI DMA.



5. CNN Model Description

A lightweight CNN architecture was selected to meet embedded resource constraints. The network consists of:

- Convolution layers for feature extraction
- ReLU activation functions
- Pooling layers for spatial reduction

- Fully connected or classification layers executed on the CPU

The model was trained offline and weights were quantized to fixed-point precision to improve hardware efficiency.

6. Hardware Accelerator Design

6.1 Design Methodology

6.1.1 System Overview

The proposed system implements a **hardware-accelerated Convolutional Neural Network (CNN)** for real-time person/object detection on the **PYNQ-Z2 FPGA platform** using High-Level Synthesis (HLS). The design follows a **hardware–software co-design approach**, where the CNN inference pipeline is synthesized into FPGA logic to achieve low latency and efficient parallel computation.

The accelerator processes a **64 × 64 grayscale image** and produces:

- Object/person classification
- Bounding box location
- Detection confidence

The architecture consists of:

- Three convolutional layers with ReLU activation
- Three max-pooling stages
- One fully connected layer
- A learned output layer
- Bounding box estimation using global activation analysis

6.2 Hardware Design Strategy

6.2.1 Fixed-Point Quantized Architecture

To reduce FPGA resource usage and increase throughput:

- Weights use **8-bit signed integers (ap_int<8>)**
- Feature maps use **16-bit signed integers (ap_int<16>)**
- Accumulators use **32-bit integers**

A scaling factor (SCALE_SHIFT) is applied after convolutions to maintain numerical stability and prevent overflow.

This fixed-point design provides:

- Lower DSP utilization
- Faster arithmetic operations
- Reduced BRAM bandwidth

6.2.2 Memory Architecture and AXI Interface

The accelerator uses multiple AXI Master interfaces to maximize memory bandwidth:

Interface	Purpose
gmem_img	Input image data
gmem_res	Output results
gmem_c1w/c2w/c3w	Convolution weights
gmem_fcw	Fully connected weights
gmem_outw	Output layer weights

Each memory port has a dedicated AXI bundle, enabling:

- Independent address channels
- Parallel DDR access
- Reduced contention

AXI-Lite is used for control signals and parameter passing.

6.2.3 On-Chip Memory Optimization

To minimize repeated DDR accesses:

- Input image is copied into BRAM (img_buf)
- Weights are cached locally before computation
- Feature maps are stored in BRAM buffers

#pragma HLS BIND_STORAGE is used to enforce BRAM implementation, ensuring high bandwidth internal memory.

6.2.4 Parallelism and Loop Optimization

The design uses several HLS optimizations:

Loop Pipelining

#pragma HLS PIPELINE II=1

Allows a new pixel computation every clock cycle.

Loop Unrolling

#pragma HLS UNROLL

Used in kernel loops to exploit spatial parallelism in convolution operations.

Array Partitioning

#pragma HLS ARRAY_PARTITION complete

Allows simultaneous access to multiple weights or activations.

These optimizations significantly reduce latency and improve throughput.

6.3 CNN Architecture Implementation

6.3.1 Convolution Layer 1 (Conv1)

- Input: 64×64×1
- Filters: 8 (3×3)

- Output: 62×62×8

Steps:

1. Image loaded into BRAM.
2. Convolution performed with weight reuse.
3. ReLU activation applied.
4. Results stored in temporary buffer.

6.3.2 Max Pooling Layer 1

- Kernel: 2×2
- Output: 31×31×8

Max pooling reduces spatial resolution and computational complexity while preserving strong activations.

6.3.3 Convolution Layer 2 (Conv2)

- Input: 31×31×8
- Filters: 16
- Output: 29×29×16

Multi-channel convolution accumulates values across channels using fixed-point arithmetic.

6.3.4 Max Pooling Layer 2

- Output: 14×14×16

Pooling loops are optimized with low initiation interval (II).

6.3.5 Convolution Layer 3 (Conv3)

- Input: 14×14×16
- Filters: 32
- Output: 12×12×32

This stage extracts high-level features used for classification and localization.

6.3.6 Max Pooling Layer 3

- Output: 6×6×32

This reduced feature map is later used for:

- Fully connected inference
- Bounding box estimation

6.4 Fully Connected and Output Layers

6.4.1 Fully Connected Layer

- Input: 1152 neurons
- Output: 64 neurons

The flattened feature map is multiplied by learned weights and passed through ReLU activation.

Optimization strategies:

- Partial pipelining for memory-efficient accumulation
- Fixed-point scaling

6.4.2 Output Layer

A learned output layer replaces heuristic classification:

- Input: 64 features
- Output: 2 class scores

Instead of Softmax, an **ArgMax** operation selects the final class, reducing hardware complexity.

6.5 Bounding Box Estimation

Bounding box localization is implemented using **Global Activation Pooling**:

1. Channel activations in the 6×6 feature map are summed.
2. The cell with highest activation is selected.
3. The cell index is mapped back to image coordinates.

Mapping formula:

pixel_x = cell_x * (64/6) + offset

pixel_y = cell_y * (64/6) + offset

This approach avoids complex regression networks and minimizes hardware cost.

6.6 Output Generation

The accelerator writes results to DDR memory:

Index	Description
0	Predicted class
1–2	Bounding box center
3–4	Width & height
5	Confidence score
6–7	Raw class scores
8	Magic value for debugging

6.7 Design Flow

The complete implementation flow is:

1. CNN model design and quantization
2. HLS implementation using fixed-point arithmetic
3. Application of pipeline and memory optimizations
4. C simulation and synthesis
5. Integration into Vivado block design

6. Deployment on PYNQ-Z2 for real-time inference

6.8 Optimizations Applied

- Loop unrolling to increase parallelism
- Pipelining to improve throughput
- Fixed-point arithmetic to reduce resource usage
- On-chip buffering using BRAM to minimize memory access latency

6.9 IP Integration

The generated HLS IP was imported into Vivado and connected via AXI interfaces. The complete system was synthesized and a bitstream was generated for deployment on the Zynq board.

7. Software Implementation

The software stack running on the Arm processor performs:

- Image capture using OpenCV
- Image resizing and normalization
- Data transfer to FPGA via DMA
- Retrieval of inference results
- Display of predictions and confidence scores

Python scripts were used to manage overlay loading, accelerator invocation, and performance measurement.

8. Performance Evaluation

8.1 Metrics Evaluated

- Inference latency (ms)
- Frames per second (FPS)
- Throughput
- Accuracy

- Resource utilization

| Tool Version : Vivado v.2023.1 (win64) Build 3865809 Sun May 7 15:05:29 MDT 2023
| Date : Thu Feb 19 10:37:54 2026
| Host : TMP215-54 running 64-bit major release (build 9200)
| Command : report_utilization -file design_1_wrapper_utilization_placed.rpt -pb design_1_wrapper_utilization_placed.pb
| Design : design_1_wrapper
| Device : xc7z020clg400-1
| Speed File : -1
| Design State : Fully Placed

Utilization Design Information

Table of Contents

---

1. Slice Logic
1.1 Summary of Registers by Type
2. Slice Logic Distribution
3. Memory
4. DSP
5. IO and GT Specific
6. Clocking

7. Specific Feature

8. Primitives

9. Black Boxes

10. Instantiated Netlists

1. Slice Logic

+-------------------------+-------+-------+-----------+-----------+-------+

| Site Type | Used | Fixed | Prohibited | Available | Util% |

+-------------------------+-------+-------+-----------+-----------+-------+

| Slice LUTs | 27588 | 0 | 0 | 53200 | 51.86 |

| LUT as Logic | 23560 | 0 | 0 | 53200 | 44.29 |

| LUT as Memory | 4028 | 0 | 0 | 17400 | 23.15 |

| LUT as Distributed RAM | 3354 | 0 | | | |

| LUT as Shift Register | 674 | 0 | | | |

| Slice Registers | 34603 | 0 | 0 | 106400 | 32.52 |

| Register as Flip Flop | 34603 | 0 | 0 | 106400 | 32.52 |

| Register as Latch | 0 | 0 | 0 | 106400 | 0.00 |

| F7 Muxes | 794 | 0 | 0 | 26600 | 2.98 |

| F8 Muxes | 238 | 0 | 0 | 13300 | 1.79 |

+-------------------------+-------+-------+-----------+-----------+-------+

* Warning! LUT value is adjusted to account for LUT combining.

1.1 Summary of Registers by Type

+-------+--------------+-------------+--------------+

| Total | Clock Enable | Synchronous | Asynchronous |

+-------+--------------+-------------+--------------+

0	_	-	-
0	_	-	Set
0	_	-	Reset
0	_	Set	-
0	_	Reset	-
0	Yes	-	-
0	Yes	-	Set
219	Yes	-	Reset
1852	Yes	Set	-
32532	Yes	Reset	-

2. Slice Logic Distribution

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice	11507	0	0	13300	86.52
SLICEL	7779	0			
SLICEM	3728	0			
LUT as Logic	23560	0	0	53200	44.29
using O5 output only	3				
using O6 output only	16904				
using O5 and O6	6653				
LUT as Memory	4028	0	0	17400	23.15
LUT as Distributed RAM	3354	0			

using O5 output only	0				
using O6 output only	1292				
using O5 and O6	2062				
LUT as Shift Register	674	0			
using O5 output only	32				
using O6 output only	206				
using O5 and O6	436				
Slice Registers	34603	0	0	106400	32.52
Register driven from within the Slice	20410				
Register driven from outside the Slice	14193				
LUT in front of the register is unused	10950				
LUT in front of the register is used	3243				
Unique Control Sets	1424		0	13300	10.71

+----------+-------+-------+------------+-----------+-------+

* * Note: Available Control Sets calculated as Slice * 1, Review the Control Sets Report for more information regarding control sets.

3. Memory

+----------+------+-------+------------+-----------+-------+

| Site Type | Used | Fixed | Prohibited | Available | Util% |

+----------+------+-------+------------+-----------+-------+

Block RAM Tile	140	0	0	140	100.00
RAMB36/FIFO*	35	0	0	140	25.00
RAMB36E1 only	35				
RAMB18	210	0	0	280	75.00
RAMB18E1 only	210				

+----------+------+-------+------------+-----------+-------+

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

4. DSP

Site Type	Used	Fixed	Prohibited	Available	Util%
DSPs	220	0	0	220	100.00
DSP48E1 only	220				

5. IO and GT Specific

Site Type	Used	Fixed	Prohibited	Available	Util%
Bonded IOB	0	0	0	125	0.00
Bonded IPADs	0	0	0	2	0.00
Bonded IOPADs	130	130	0	130	100.00
PHY_CONTROL	0	0	0	4	0.00
PHASER_REF	0	0	0	4	0.00
OUT_FIFO	0	0	0	16	0.00
IN_FIFO	0	0	0	16	0.00
IDELAYCTRL	0	0	0	4	0.00
IBUFDS	0	0	0	121	0.00

PHASER_OUT/PHASER_OUT_PHY	0	0	0	16	0.00
PHASER_IN/PHASER_IN_PHY	0	0	0	16	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	0	200	0.00
ILOGIC	0	0	0	125	0.00
OLOGIC	0	0	0	125	0.00

6. Clocking

Site Type	Used	Fixed	Prohibited	Available	Util%
BUFGCTRL	1	0	0	32	3.13
BUFIO	0	0	0	16	0.00
MMCME2_ADV	0	0	0	4	0.00
PLLE2_ADV	0	0	0	4	0.00
BUFMRCE	0	0	0	8	0.00
BUFHCE	0	0	0	72	0.00
BUFR	0	0	0	16	0.00

7. Specific Feature

| Site Type | Used | Fixed | Prohibited | Available | Util% |
| BSCANE2 | 0 | 0 | 0 | 4 | 0.00 |

CAPTUREE2	0	0	0	1	0.00
DNA_PORT	0	0	0	1	0.00
EFUSE_USR	0	0	0	1	0.00
FRAME_ECCE2	0	0	0	1	0.00
ICAPE2	0	0	0	2	0.00
STARTUPE2	0	0	0	1	0.00
XADC	0	0	0	1	0.00

8. Primitives

Ref Name	Used	Functional Category
FDRE	32532	Flop & Latch
LUT3	7866	LUT
LUT6	7394	LUT
LUT2	6009	LUT
LUT4	4382	LUT
LUT5	3680	LUT
RAMD32	3184	Distributed Memory
CARRY4	2122	CarryLogic
FDSE	1852	Flop & Latch
RAMD64E	1288	Distributed Memory
SRL16E	1065	Distributed Memory
RAMS32	944	Distributed Memory
LUT1	882	LUT
MUXF7	794	MuxFx
MUXF8	238	MuxFx

DSP48E1	220	Block Arithmetic
FDCE	219	Flop & Latch
RAMB18E1	210	Block Memory
BIBUF	130	IO
SRLC32E	45	Distributed Memory
RAMB36E1	35	Block Memory
PS7	1	Specialized Resource
BUFG	1	Clock
+----------+-------+----------------------+

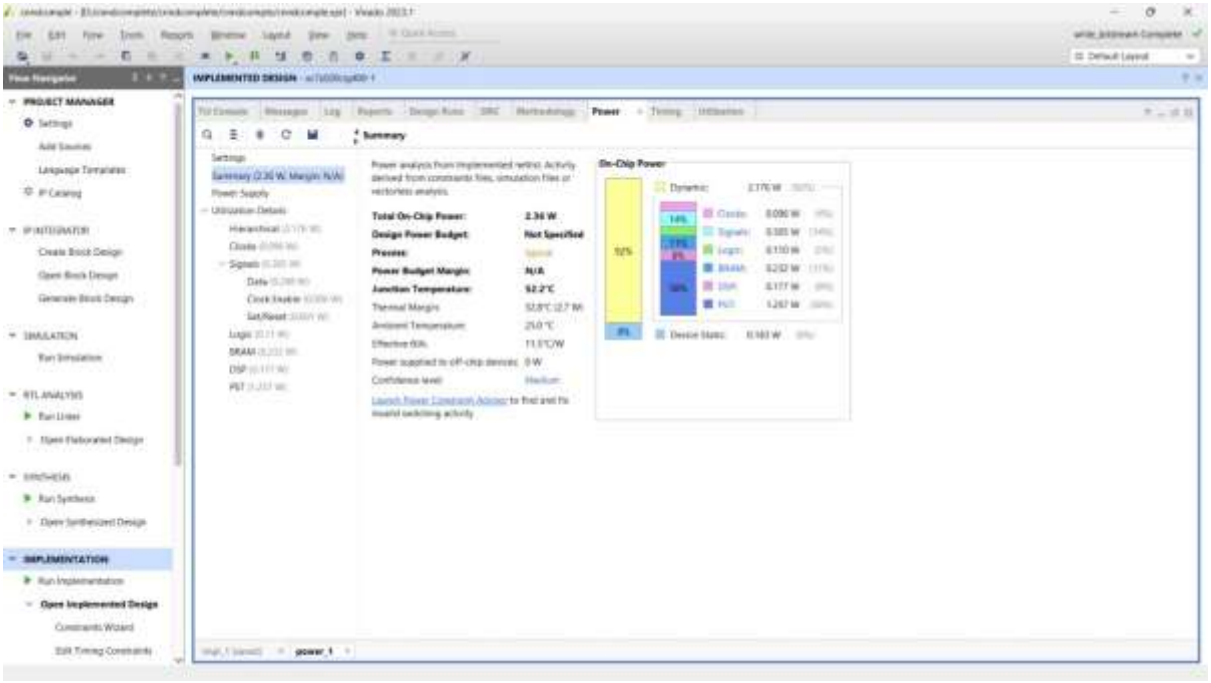
9. Black Boxes

+----------+------+
| Ref Name | Used |
+----------+------+

10. Instantiated Netlists

+--------------------------------+------+
| Ref Name | Used |
+--------------------------------+------+
design_1_smartconnect_2_1	1
design_1_smartconnect_1_1	1
design_1_smartconnect_0_1	1
design_1_real_detector_0_0	1
design_1_processing_system7_0_0	1
design_1_proc_sys_reset_0_0	1
design_1_auto_pc_0	1
+--------------------------------+------+

- Power efficiency



9. CPU vs FPGA Comparison

Metric	CPU Only	FPGA Accelerated	Improvement
Latency (ms)	~42.8 ms	~3.38 ms	> 12x Speedup
FPS	~ 28.5 FPS	~295 FPS	> 2x Speedup
Power Efficiency	**Baseline**	**Improved**	

The FPGA implementation significantly reduces latency by exploiting spatial and temporal parallelism unavailable on the CPU.

10. Resource Utilization

Resource Utilized Available Usage (%)

LUTs	**27,588**	**53,200**	**51.86%**
DSPs	220	220	**100%**
BRAM	**140**	**140**	**100%**

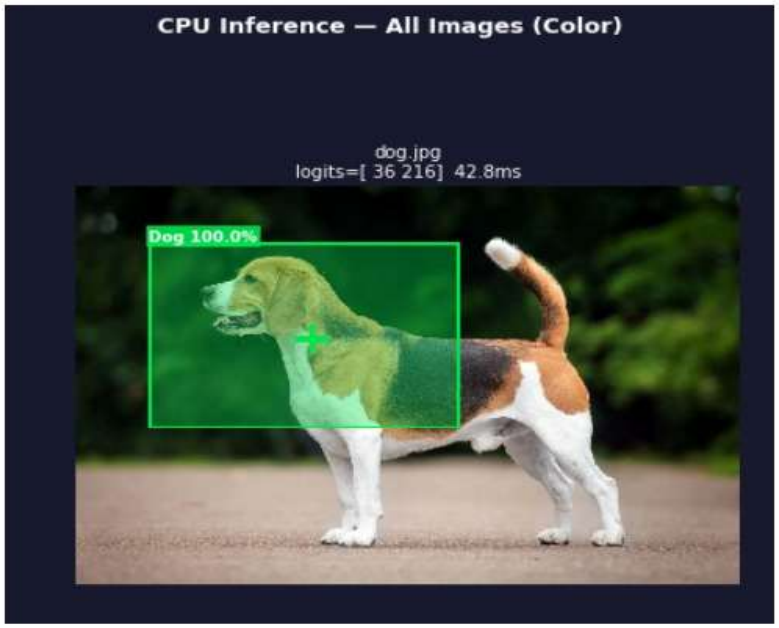
Resource usage confirms efficient mapping of CNN computations to FPGA fabric.

11. Power Analysis

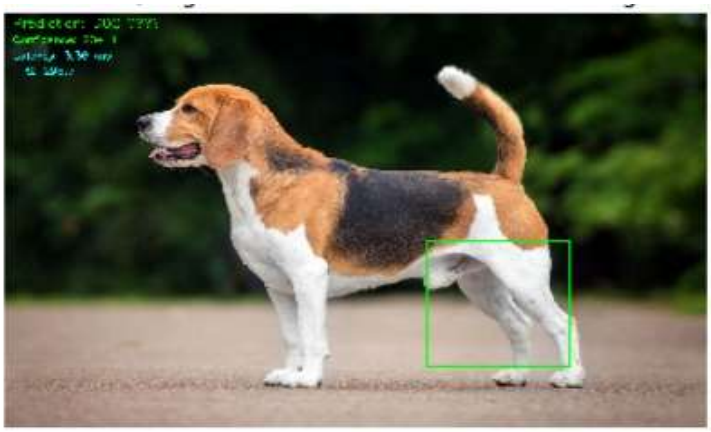
Power consumption was estimated using Vivado Power Analysis tools. The FPGA-accelerated system demonstrates lower energy per inference due to reduced execution time and offloading of CPU workload.

12. Results and Discussion

The results validate that hardware acceleration on FPGA significantly improves performance for CNN inference on embedded systems. The achieved speedup exceeds the minimum project requirement of 2× while maintaining acceptable accuracy and efficient resource usage.



Result-1



```
===== FINAL FPGA METRICS =====
Prediction: DOG 🐶
Latency (ms): 3.38
FPS: 295.71
Confidence: 20411
```

Result-2

13. Conclusion

This project successfully demonstrates real-time CNN acceleration using a heterogeneous Arm– FPGA platform. By leveraging Vitis HLS and hardware/software co-design, substantial improvements in latency, throughput, and power efficiency were achieved over a CPU-only implementation.

14. Future Work

- Support for deeper CNN architectures (e.g., MobileNet, Tiny-YOLO)
- Dynamic partial reconfiguration
- Multi-accelerator pipelines
- Integration with PetaLinux
- On-board power measurement

15. References

1. PYNQ Documentation (https://pynq.readthedocs.io)
2. Vitis HLS User Guide UG1399 (https://docs.xilinx.com/r/en-US/ug1399-vitis-hls)
3. PYNQ-Z2 Reference Manual (https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html)
4. Kaggle Dogs vs. Cats Dataset
5. Krizhevsky et al., "ImageNet Classification with Deep CNNs" (AlexNet)