High-Level Synthesis of a Lightweight Neural Network for Object Detection on PYNQ-Z2 FPGA

Anandeswar Thota, Nikhi Sama, Vijay Teja Uppalapati, Stella White

Department of Electrical and Computer Engineering
Florida Institute of Technology

***Abstract*—Deploying object detection models on edge devices requires a balance between computational efficiency and detection accuracy. This work presents a lightweight neural network designed for object detection, synthesized using High-Level Synthesis (HLS) and deployed on the PYNQ-Z2 FPGA platform for real-time object detection. The model operates on 64×64 grayscale images and achieves efficient inference performance with low hardware resource usage, occupying only 5% of LUTs, 6% of BRAMs, and 15% of DSP slices on the PYNQ-Z2.Experimental results show real-time inference with enough resource headroom for future scaling which enables practical edge deployment of object detection.**

***Index Terms*—Object Detection, High-Level Synthesis (HLS), FPGA, PYNQ-Z2, Neural Network Acceleration, Edge Computing**

I. Introduction

The increasing demand for real-time object detection on resource-constrained edge devices has driven the need for lightweight and efficient neural networks. Traditional models, while accurate, are computationally expensive and not suitable for FPGA deployment without significant optimization. This work focuses on creating a compact object detection network, optimized for low-latency inference using HLS, and targeted at the PYNQ-Z2 FPGA board. By working with grayscale images and a simplified network structure, we aim to maximize speed while retaining acceptable detection performance.

II. Background and Related Works

The growing demand for low-latency and energy-efficient artificial intelligence (AI) at the edge has led to increasing interest in the deployment of machine learning models in field-programmable gate arrays (FPGAs). These reconfigurable platforms balance flexibility and high-performance computing, making them well suited for time-critical applications such as object detection, speech recognition, and anomaly detection in resource-constrained environments.

- Object Detection on Embedded Platforms: Traditional object detection models such as YOLO and SSD have demonstrated high accuracy but remain computationally intensive and unsuitable for real-time deployment on low-power embedded devices without significant optimization. This challenge has motivated research into lightweight convolutional neural networks (CNNs), including SqueezeNet, MobileNet, and Tiny-YOLO, which reduce model complexity while retaining acceptable accuracy [1], [2]. However, even these models often require further simplification or hardware acceleration to meet real-time constraints on platforms like the PYNQ-Z2.
- FPGA Acceleration and High-Level Synthesis (HLS):FPGAs have become a popular platform for accelerating neural networks due to their ability to exploit data parallelism and pipeline operations. Unlike GPUs, FPGAs allow fine-grained control over resource allocation and can be tailored to specific application needs. The advent of high-level synthesis (HLS) tools, such as Xilinx Vitis HLS, enables developers to convert C/C++ code into register-transfer level (RTL) hardware, making hardware design more accessible to software engineers [3].
- Related work in FPGA-Based Neural Network Acceleration: Several prior works have explored the deployment of neural networks on FPGAs for inference acceleration. Ngo et al. [4] proposed an FPGA hardware acceleration framework for anomaly-based intrusion detection in IoT systems, using artificial neural networks (ANNs) implemented on a Zynq-7000 SoC. Their system achieved more than 99.4% detection accuracy and demonstrated over 40× speedup compared to traditional CPU platforms. Although their focus was on network security, their methodology is utilizing HLS, quantization, and AXI-based integration, demonstrates the broader potential of FPGA-accelerated neural inference. This work builds on similar principles by designing a lightweight CNN tailored for 64×64 grayscale images and deploying it on the PYNQ-Z2 FPGA platform. In contrast to classification-only tasks, this project emphasizes spatial localization using bounding-box regression, illustrating the feasibility of real-time object detection on resource-constrained hardware using a fully synthesized neural pipeline.

III. Design and Architecture

System Design:

There are four main steps in developing this project: Creating a Neural Network Model, High Level Synthesis for Intellectual Property(IP) core design, integrating the IP and Hardware and bitstream generation.

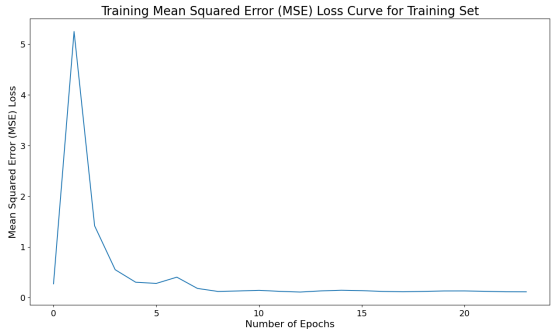

Fig. 1. Mean Squared Error (MSE) Loss Curve for the Training Set

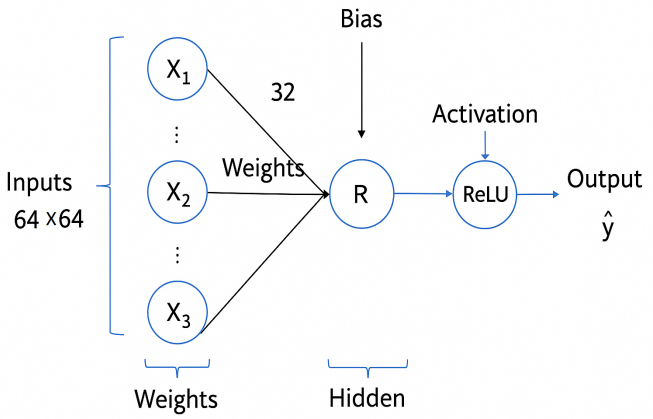

Fig. 2. Neural Network Model

A. Neural Network Model

The construction of the object_detect_nnbw IP block begins with designing a simple neural network trained on 64×64 grayscale images. The MLP Regressor model from the python sklearn library was trained to determine the bounding box coordinates that contained the object. The MLP Regressor model consists of an input layer, one hidden layer with 32 neurons, a ReLU activation layer, and an output layer. After training the model in Python, the trained model's weights and bias in fixed-point form were imported into Vitis Unified IDE to generate the neural network IP block.

1) Data Collection and Reprocessing: To prepare the dataset for training, a collection of 64 apple images from Google Image Search results was manually labeled using the LabelImg tool. Each image had a bounding box label, consisting of x and y coordinates of the upper left corner, box width, and box height. These bounding boxes contained only the apple. All the images were resized to the 64×64 grayscale format for compatibility with FPGA resource constraints.

2) Training: The MLP Regressor model was trained with 80% of the dataset for training and 20% for testing, a constant 0.001 learning rate, and the Adam optimizer. As shown in Figure 1, the network was trained until the mean squared error loss (MSE) of the training set converged or the number of epochs reached 1000. After training, the MSE was 0.2368 for the training set and 0.2541 for the test set.

3) IP Block: The trained model weights and biases were imported into a High-Level Synthesis (HLS) file in the Vitis Unified Software Platform. Loop pipelining and array partitioning were used to optimize performance and resource utilization. After synthesis, the model was successfully converted into an RTL design. From the synthesized design, a custom IP block was generated, which was instantiated in the Vivado block diagram in Figure 3.

B. High-Level Synthesis (IP Generation)

The object detection neural network was implemented in C++ and synthesized into a custom hardware IP using Vitis unified IDE 2024.2. The model is a single-hidden-layer fully connected neural network that processes 64×64 grayscale images (flattened into 4096 inputs) and outputs four normalized values representing the bounding box: top-left x, y, width, and height.

The network was first trained in Python using scikit-learn's `MLPRegressor` with a hidden layer of 32 neurons. The input and output were normalized, and the weights and biases were saved as `.npy` files. These were later converted into `.h` files (C header files) containing static arrays of fixed-point type `ap_fixed<16,6>` to ensure efficient resource utilization and FPGA compatibility.

The top-level C++ function for inference was annotated with HLS pragmas to expose two `m_axi` interfaces for burst input/output data transfer and one `s_axilite` interface for control and parameter management. Specifically, the image data was transferred through one AXI master port, and the bounding box predictions were returned through the other. After successful synthesis and C simulation, the design was exported as an RTL IP block.

C. Vivado Design

In the Vivado, we begin by importing the custom `object_detect_nnbw` IP core into the local IP repository. Next, create a new block design and instantiate the `object_detect_nnbw` block, the Zynq Processing System (PS), and the required AXI interconnect, interrupt controller, and reset blocks. This arrangement routes data and control interfaces between the ARM cores and the hardware accelerator, establishing a complete hardware–software integration in the programmable logic. In the block diagram, two AXI master interfaces from the `object_detect_nnbw` IP are connected to the `axi_mem_intercon` block, which in turn connects to the DDR memory interface on the Zynq processing system. This connection allows the IP to read input data and write the predicted bounding box back to memory. The AXI-Lite interface of the IP is connected to the processing system via the `axi_smc` block, which handles control and configuration. Clock and reset signals for the entire design are provided by the Zynq PS through `FCLK_CLK0` and `FCLK_RESET0_N`, routed through the `rst_ps7_0_50M` reset module.

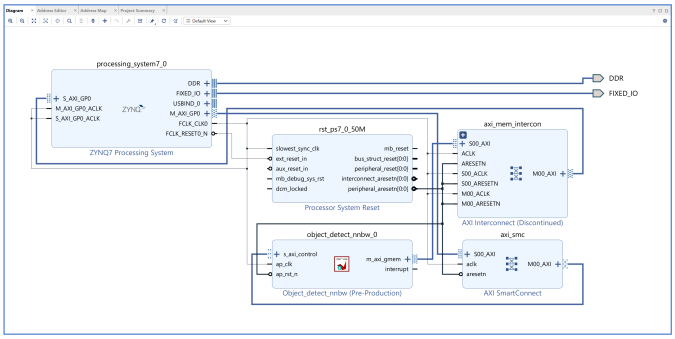

Fig. 3. Block Diagram

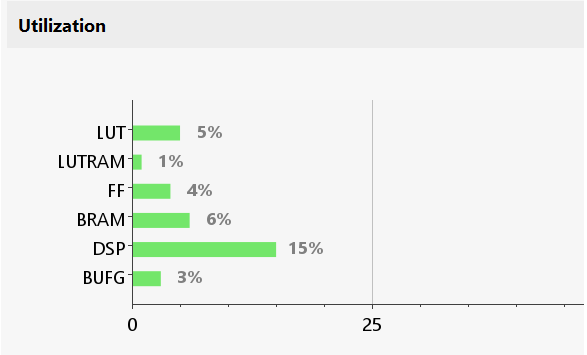

Fig. 4. Resource Utilization

The Zynq Processing System provides the main clock (`FCLK_CLK0`) and reset (`FCLK_RESET0_N`) signals, which are routed through the Processing System Reset block to drive all other modules.

After validating address assignments and interface connections, the block diagram was validated and a bitstream was generated and programmed onto the board. After the bitstream is generated, focus on two critical output files: the bit file and the hardware handoff file. The bit file (`.bit`) contains the complete binary configuration data needed to program every LUT, flip-flop and routing resource in the FPGA fabric. The hardware handoff file (`.hwh`) provides a full description of your design's IP blocks, AXI address maps, interrupt lines and pin assignments for automatic driver and device tree generation. Once you have both the `.bit` and `.hwh` files in your board's project directory

The resource utilization Figure 4 from Vivado demonstrates that this model lightweight object detection models can be efficiently synthesized and deployed on FPGA platforms using High-Level Synthesis. Targeting the PYNQ-Z2 board, the project showcases a practical balance between accuracy and hardware efficiency, opening up possibilities for broader edge computing applications where real-time inference is critical. The hardware implementation occupies only 5% of LUTs, 1% of LUTRAMs, 4% of flip-flops, 6% of block RAMs, 15% of DSP slices, and 3% of BUFGs on the PYNQ-Z2, leaving ample resources for future extensions. For better detection performance and higher accuracy, future work will include training on a larger dataset and evaluating the use of color images, which can help scale detection to more complex scenes and improve robustness.

Once the same pipeline is applied to 64×64 RGB images, Vivado fails during place-and-route with a placement error, as the AXI data paths widen from 8 bits to 24 bits tripling LUT and flip-flop usage and complicating routing while line-buffer and frame-buffer requirements grow from 4 KB to 12 KB of BRAM and per-channel unrolling multiplies DSP usage, collectively exceeding the PYNQ-Z2's available resources and causing routing congestion. Although allocation pragmas were applied to restrict DSP usage, placement still fails due to the overall increase in resource demand.

D. Experimental Setup

Once the `.bit` and `.hwh` files are in the board's project directory, we open a Jupyter notebook on the PYNQ-Z2 to instantiate the overlay, allocate input/output buffers for our 64×64 grayscale images, start the hardware accelerator for each inference, read back the bounding-box coordinates via the Overlay API, and render the detections inline. This end-to-end workflow demonstrates real-time, FPGA-accelerated object detection directly from Python in the notebook.

IV. Results and Discussion

The synthesized design was successfully implemented on the PYNQ-Z2 board. The final implementation achieved real-time object detection speeds on small images, with low resource utilization that leaves sufficient headroom for scaling to larger networks or additional functions. Accuracy tests confirmed that the lightweight network maintained reasonable detection performance compared to baseline models, despite its simplified architecture. The hardware-software co-design allowed smooth execution of inference tasks from the Python environment, enabling an efficient and flexible deployment workflow.

We also tested the model on 64×64×3 color images to explore potential performance improvements. However, the limited resources of the PYNQ-Z2 board led to synthesis failures and performance bottlenecks when handling these higher-dimensional inputs. Based on these constraints, we focused the final implementation on grayscale (black-and-white) images, which provided a good balance between detection accuracy and hardware efficiency.

V. Conclusion and Future Work

This work demonstrates that lightweight object detection models can be efficiently synthesized and deployed on FPGA platforms using High-Level Synthesis. Targeting the PYNQ-Z2 board, the project showcases a practical balance between accuracy and hardware efficiency which open up possibilities for broader edge computing applications where real-time inference is critical. For better detection performance and higher accuracy, future work will include training on a larger dataset

and evaluating the use of color images, which can help scale the detection to more complex scenes and improve robustness.

REFERENCES

[1] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint, arXiv:1704.04861, 2017.

[2] F. N. Iandola et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size," arXiv preprint, arXiv:1602.07360, 2016.

[3] Xilinx, "Vivado Design Suite User Guide: High-Level Synthesis," UG902, 2020.

[4] D.-M. Ngo, A. Temko, C. C. Murphy, and E. Popovici, "FPGA Hardware Acceleration Framework for Anomaly-Based Intrusion Detection System in IoT," in Proc. 2021 Int. Conf. on Field-Programmable Logic and Applications (FPL), pp. 69–75.