FPGA based Hardware Implementation of Deep Learning Models for Target Identification

Aryan Shah[1], Mantra Solanki[2], Vaishali Dhare[3]
[1,2,3]Institute of Technology, Nirma University, Ahmedabad, India
21bec112@nirmauni.ac.in, 21bec121@nirmauni.ac.in, vaishali.dhare@nirmauni.ac.in

***Abstract*—Performance is one of the key criteria for the target detection algorithm. Although the Deep Neural Network (DNN) models used for target detection have achieved performance parameters such as accuracy and speed, the fastest processing is required for critical applications. It is well proven that hardware is the fastest compared to software in terms of data processing. Therefore, it is need of the day to deploy the target detection algorithms on a hardware. In this paper, the DNN based target detection algorithm is deployed on a Field Programmable Gate Array (FPGA). It is implemented using the DNN Development Kit (DNNDK) and deployed on PYNQ-Z2 FPGA using the Xilinx platform. In this paper, yield flexibility, efficiency in power usage, and fast processing time are achieved compared to Graphics Processing Units (GPUs). The target detection models YOLOv3 and tiny YOLOv3 are deployed on PYNQ-Z2. The performance parameters of the proposed work, namely power, accuracy, and speed, are compared with the T4 GPU, RTX 2028 Ti, and Jetson Nano.**

***Index Terms*—Deep Learning, DNNDK, FPGA, Machine Learning, Object Detection, YOLO**

I. Introduction

Machine Learning (ML) has transformed computer vision by allowing machines to interpret, understand, and analyze any type of data. Deep learning models such as YOLO, CNNs, MobileNet, ResNet, etc. have been widely used for applications such as object detection, recognition, and segmentation [1]. However, real-time applications require the system to have low latency, better processing, and energy efficient. GPUs deliver high computational power, but come with a trade-off between power consumption and computational capabilities. Additionally, scalability and adaptability for tasks-specific applications are limited in fixed-architecture hardware.

The CPUs and GPUs have been the traditional platforms for training and deploying deep learning models but they face significant limitations, especially when real-time or embedded applications. The primary issues with existing solutions include:

- Though powerful, GPUs are known their high power consumption, making them unsuitable for low-power, real-time applications, such as autonomous drondrones,vices and mobile systems.
- Traditional CPUs and GPUs are designed for general-purpose computing, and while they are efficient at han-For deep learning, though, they often struggle with real-time performance because of their lack of specialization in particular tasks. This results in higher latency, which is a critical limitation in applications like autonomous driving or medical diagnostics, where a decision needs to be made as quickly as possible.
- GPUs and CPUs have fixed architectures, which means they cannot be tailored or optimized for the handling of specific deep learning models. Therefore, they are often inefficient when deployed for highly specialized tasks, where customized processing units could significantly improve performance.
- Because of the immense size and raising complexity of deep learning models It tends to grow; scaling these models on CPUs and GPUs becomes increasingly difficult. In particular, GPUs face limits in memory bandwidth and computational throughput, leading to bottlenecks during model execution. These challenges have led to the exploration of alternative Hardware solutions offering low power consumption lower latency, and better adaptability for specialized deep learning tasks.

On the other hand, FPGAs provide a promising solution by offering customization and adaptability, parallel processing, and low power consumption. FPGAs can be configured for specific ML models, allowing real-time inference with low latency and reduced power consumption compared to GPUs and CPUs. In this paper, DNN models for object detection are deployed on FPGA. It is configured to make it well-suited for performance, latency, and power consumption. The performance parameters of object detection using FPGA are compared with the GPU.

Earlier, PYNQ framework, that supports a Python-based hardware/software codesign environment to perform CNN inference for object recognition on Xilinx FPGA was demonstrated in [2]. In [3], FPGA-based general YOLO model, tailored for object detection and classification in edge computing devices are is proposed. In [4] , CNN implementation on FPGAs is presented. It focuses on model pruning and quantization with the aim of increasing performance. It achieved impressive latency and power reduction with almost no loss in accuracy and became the benchmark for the latest generation of FPGA-based deep learning. However, even with the advancement, challenges persist in FPGA-based systems: limited resources restrict large-model deployment, and the development complexity is very high due to specialized hardware programming. Besides, FPGAs are less suited for training neural networks, and normally, a hybrid workflow is needed, where training should be done on GPUs, and for inference, FPGAs are used. Limitations with FPGAs have again brought up the need for

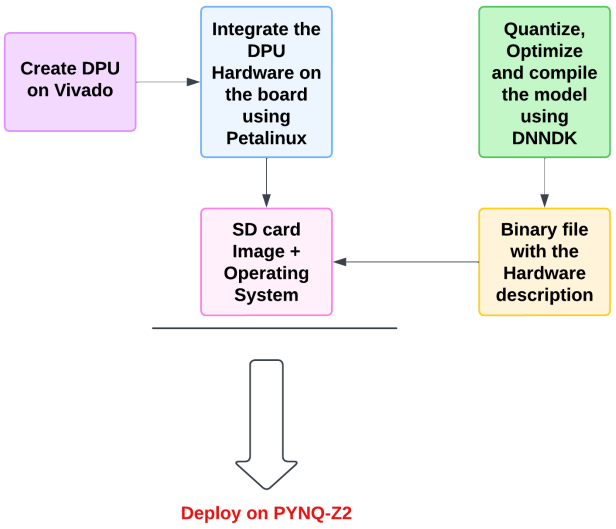

Fig. 1. Process Flow

further innovation in performance, efficiency, and usability of hardware-accelerated AI/ML solutions. In this paper, we tried to address the inefficiency of traditional processors in running neural networks by exploring FPGA implementations. This is typically achieved through hardware-specific optimizations, including pipelining and reconfigurable logics that exploit intrinsic parallelism inherent in FPGAs. Ease of deployment, provided by toolchains like Xilinx DNNDK and Vitis AI, enables their efficient execution on resource-constrained devices for the compressed and quantized neural networks. Further, the performance parameters of various models are compared with different paltforms.

The rest of the paper is organized as follows: Section II presents the process flow. Implementation of hardware is presented in Section III. The results and comparisons of performance parameters with available methods are presented in Section IV. The paper is concluded in Section V.

II. Process Flow

The process of implementing deep learning models, particularly for target detection on a FPGA involves several stages. Each step is critical for optimizing performance, reducing power consumption, and ensuring real-time inference. This Section outlines the major steps involved in the process flow for deploying deep learning models onto a FPGA, more specifically using platforms like Xilinx's PYNQ-Z2 board.

A. Model Selection and Preprocessing

The first step involves selecting an appropriate deep learning model for the particular task at hand, say object detection. During this process, a YOLO model is chosen due to its high precision and speed in the detection of multiple objects in real-time.

- **Model Selection:** A pre-trained YOLO model, trained on a dataset such as COCO or ImageNet, is chosen due to its capability to perform fast object detection by predicting bounding boxes and class probabilities directly from full images in one pass through the network.
- **Dataset Preparation:** The dataset used for training and validation must be prepared in a format compatible with the deep learning framework of choice (such as TensorFlow or PyTorch). This involves annotating images with object labels and bounding boxes and splitting the dataset into training and validation sets.
- **Model Preprocessing:** Before deployment, the model may need to be preprocessed to convert it into a compatible format for target hardware. This includes resizing input images, normalizing pixel values, and converting the model architecture into a framework like TensorFlow's frozen graph format.

B. Network Compression and Quantization

Given the resource constraints of embedded systems and FPGAs, deep learning models must be optimized for efficient execution. This optimization is typically achieved through compression and quantization.

- **Model Compression:** Large models with many layers and parameters are compressed to fit the limited hardware resources available on the FPGA. Techniques such as pruning (removing inessential links between the neural network) are used to reduce the size of the model without significantly impacting its accuracy.
- **Quantization:** In quantization, the model's 32-bit floating-point weights and activations are converted to lower precision formats, such as 8-bit integers, to reduce memory bandwidth and computational complexity. Tools like Xilinx's DECENT Deep Compression Tool are used to perform the quantization while maintaining the model's accuracy. This step is crucial since FPGAs have limited precision and require models to fit within the hardware's operational capabilities.
- **Calibration:** A small calibration dataset is used during the quantization process to fine-tune the model and ensure it performs well with reduced precision weights.

C. Compilation for FPGA Deployment

After quantization, the next step is to compile the optimized model into a binary format that the FPGA can execute. This process requires dedicated tools like Xilinx's DNNDK (Deep Neural Network Development Kit) and Vitis AI.

- **Model Compilation:** The quantized model is passed through DNNDK's Deep Neural Network Compiler (DNNC), which translates the neural network architecture into hardware-specific instructions that the FPGA can process. The compiler maps the model's layers to the Deep Learning Processing Unit (DPU) on the FPGA, balancing workload distribution across computational resources.
- **DPU Configuration:** The FPGA's Deep Learning Processing Unit (DPU) is configured based on the model's

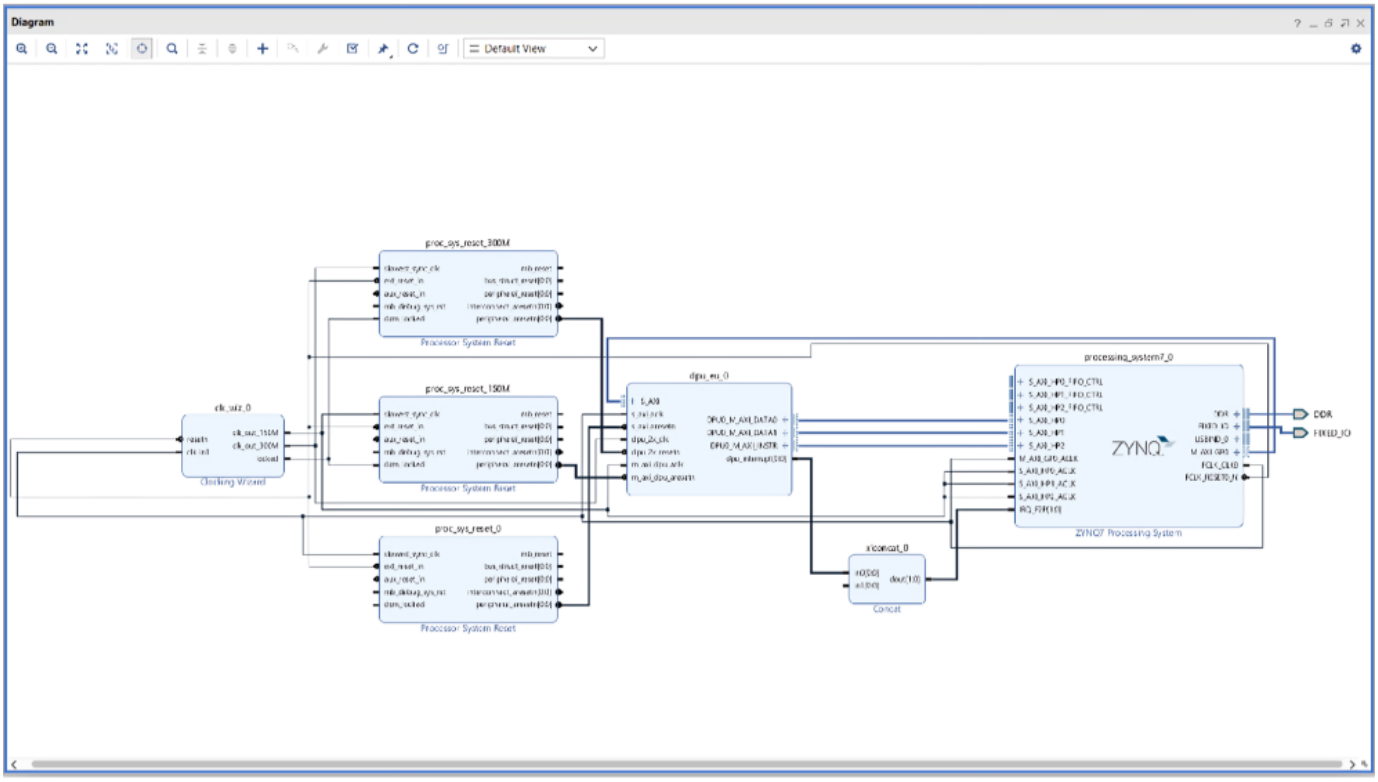

Fig. 2. Interconnection between Xilinx DPU and the PYNQ Z2.

architecture. Parameters such as the number of processing cores, memory block size, and channel parallelism are specified to ensure the FPGA is fully optimized to run the model. This configuration ensures that hardware resources such as logic gates and DSP (Digital Signal Processing) slices are efficiently utilized.

- **Binary Generation:** Once compiled, the output of DNNC is a binary file (an ELF file) containing instructions for the model's execution on the FPGA. This binary file is deployed on the FPGA along with necessary runtime components.

III. Hardware Implementation

A. Description of the Design Environment

The hardware implementation involves the Vivado Design Suite and the Xilinx PYNQ-Z2 board. Vivado enables high-level synthesis, simulation design, and the integration of multiple hardware modules, while deployment is executed on the PYNQ-Z2 board. This board integrates an SoC from the Zynq-7000 series, pairing an ARM Cortex-A9 processor for general-purpose processing with programmable logic for hardware acceleration. This combination makes deep learning processing highly feasible on this platform.

B. PL-PS Configurations and DPU Design

The Programmable Logic (PL) is integrated with the Processing System (PS) using AXI interconnects. AXI-HP slave interfaces are employed for efficient data transfers between the PL and PS, with the Direct Memory Access (DMA) minimizing CPU involvement in these transfers. To further enhance data transfer rates and reduce latency, the DMA operates in burst mode.

A Deep Learning Processing Unit (DPU) is implemented in the programmable logic to accelerate deep learning inference. The DPU is optimized for computation-intensive operations such as convolutions, pooling, and activations, enabling direct processing of neural network layers within the FPGA. This reduces inference time compared to traditional CPU or GPU implementations. Key features of the DPU include:

1) **Number of Cores:** Multiple cores are configured to enhance parallel processing, increasing throughput and reducing processing time.
2) **Memory Usage - BRAM and UltraRAM:** On-chip memory is configured to store weights, activations, and intermediate data efficiently, minimizing access to external memory.
3) **Parallel Input Channels and Pixels:** The DPU processes multiple input channels or pixels in parallel, maximizing resource utilization and speeding up convolution operations.
4) **Advanced Neural Network Operations:** The DPU optimizes operations such as depthwise separable convolutions, max pooling, and ReLU activations.
5) **DSP Slice Utilization:** Matrix multiplication tasks and other operations requiring high-performance computation are executed on the DSP slices, ensuring efficient deep

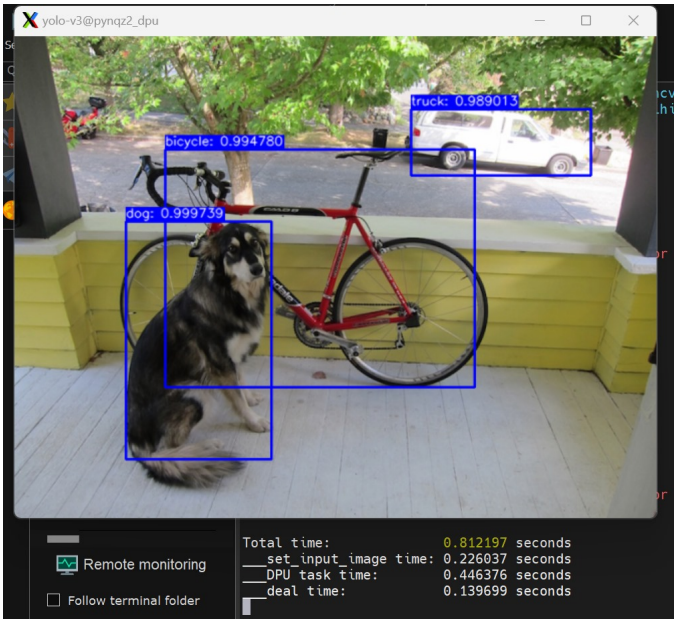

Fig. 3. Inference

learning computation.

This configurability allows the DPU to handle lightweight models for power-sensitive applications and complex models for high-performance scenarios.

In our design, the IRQ_F2P interface is utilized for interrupt handling, enabling effective communication between the PL and PS for real-time responses. The PS coordinates the flow and manages tasks triggered by the PL. DMA supports high-speed, low-overhead transfers between memory and the DPU, relieving the CPU from repetitive data handling tasks.

By leveraging these optimizations, the system achieves low-latency and energy-efficient inference, demonstrating the potential of FPGAs in AI/ML applications for real-time target identification. A well-configured DPU, combined with efficient memory usage and optimized data handling mechanisms, ensures robust performance in resource-constrained environments.

IV. Performance Evaluation

The YOLOv3 model deployed on the PYNQ-Z2 FPGA results in a very fast inference process, capable of detecting objects and generating bounding boxes in less than one second. This performance makes the model highly suitable for real-time applications. However, such performance should be analyzed in comparison with a more standard computing platform to understand the relative benefits and trade-offs.

For all the tests, the same input image was used to ensure a fair and objective comparison of results. A commonly used image that contains a dog, a bicycle, and a pickup truck was selected. This choice kept the computational load constant and avoided variability in computational requirements due to differences in image complexity, such as pixel density or object detail.

The performance of the implemented system was evaluated using the following key metrics:

- **Accuracy (mAP):** Mean Average Precision measures the effectiveness of the model in detecting and classifying targets.
- **Inference Time:** Time taken for the system to process a single input and produce predictions.
- **Execution Time:** Total time for pre-processing, inference, and post-processing.
- **Power Consumption:** Energy required during inference, crucial for evaluating efficiency in embedded applications.

A. Comparison with other platforms

The experiments were carried out using YOLOv3 and Tiny YOLOv3 on four platforms: PYNQ-Z2 FPGA, NVIDIA T4 GPU, NVIDIA RTX 2080 Ti GPU, and Jetson Nano. The results are summarized in Table I.

TABLE I
Performance Comparison Across Platforms

Model/Platform	PYNQ-Z2	T4 GPU	RTX 2080 Ti	Jetson Nano
YOLOv3				
Total Time (ms)	814	1673 (1st)	30	1000
Inference Time (ms)	446	100	17	500
FPS	2	10	60	2
Power Consumption	Low	High	High	Low
Tiny YOLOv3				
Total Time (ms)	294	267 (1st)	20	800
Inference Time (ms)	42	83	12	400
FPS	5	12	83	3
Power Consumption	Low	High	High	Low

The NVIDIA T4 GPU was tested in Google Colab and had a higher FPS than the PYNQ-Z2 FPGA due to its high performance architecture. On the other hand, for PYNQ-Z2, the inference time corresponds to the DPU task time, which is the time needed for the DPU to make a prediction. The total execution time of PYNQ-Z2 also considers ”Set Input Image Time”-it is the time for the input image to be captured by the FPGA-and ”Deal Time,” which pertains to the time it draws the bounding box for the detection results. That contributes a bit to the increase in the total processing time when performed on FPGAs.

Thus, it demonstrates how a small form factor of such a PYNQ-Z2 FPGA consistently had very low power consumption and could therefore easily be used with virtually any energy-sensitive, real-time application that needs an edge. Similarly, to do that, much better speed is available with the inference of the NVIDIA T4 GPU and RTX 2080 Ti, but with increased consumption of high voltage, its employment in such setups is restricted. Jetson Nano, though highly power-efficient, is bottomed in both pure speed and overall computation capabilities.

This effectively balances performance against power efficiency on all FPGAs, including PYNQ-Z2, although for lightweight models, as in the Tiny YOLOv3 case. Thereafter, such hardware could thus be used very well in different scenarios of embedded real-time applications-a factor to be considered, especially in reaching real-time factors. Future improvements in

TABLE II
COMPARISON OF OBJECT DETECTION PERFORMANCE WITH EXISTING WORKS

Model/Platform	NN Model	Image Size	FPS	Power (W)	Efficiency (FPS/W)	Inference Time (ms)	Total Time (ms)	Accuracy (%)
This Work (PYNQ-Z2)	YOLOv3	416×416	2	Low	-	446	814	96
	Tiny YOLOv3	416×416	12	Low	-	42	294	90
Wang et al. (Spartan-6) [1]	CNN	32×32	16	0.79	20.25	-	-	96
Heller et al. (Kria KV 260) [2]	YOLOv4 Tiny	HD	15	8	1.87	-	-	75
Corcoran et al. (VCU110) [3]	YOLOv3 Tiny	416×416	69	15.4	4.5	-	-	-
Nguyen et al. (ZCU104) [4]	YOLOv4 Tiny	HD	125	26.4	4.7	-	-	78
Valadanzoj et al. (ZC706) [5]	YOLOv4 Tiny	416×416	55	13.6	4	-	-	79
Amin et al. (Kria KV260) [6]	YOLOv3 Tiny	HD	15	3.5	2.2	-	-	99

hardware can further make such FPGA platforms at least competitive, even in more computationally intensive applications that require high speed.

B. Comparison with Existing works

To assess the performance of the suggested system, a comparison was drawn with other FPGA-based object detection models in recent literature. Here, the analysis is focused on key metrics such as frames per second (FPS), inference time, power consumption, and accuracy to identify the trade-offs in computational efficiency versus resource utilization. Table II shows the summary of the comparison results.

The results indicate that while some implementations, such as Nguyen et al. and Corcoran et al., achieve higher throughput (125 FPS and 69 FPS, respectively), it is at the cost of significantly higher power consumption (26.4W and 15.4W, respectively). In contrast, the proposed PYNQ-Z2-based implementation maintains a low power profile, making it more suitable for power-efficient embedded applications. Additionally, our model achieves a competitive accuracy of 96% for YOLOv3 and 90% for Tiny YOLOv3, which is comparable or superior to other FPGA-based implementations, including Wang et al. (96%) and Amin et al. (99%).

The system proposed here spends an inference time of 446 ms for YOLOv3 and 42 ms for Tiny YOLOv3, with total execution times of 814 ms and 294 ms, respectively. While other FPGA-based models report shorter inference times, their success typically comes at the cost of higher power consumption and more complex hardware architectures. The trade-off of our system between processing speed and power consumption positions it favorably for real-time applications in resource-constrained environments.

Although some high-performance FPGA designs exist with more FPS than our system, the PYNQ-Z2 model realizes a good balance between power efficiency, accuracy, and processing time. This makes it extremely suitable for applications that require real-time performance and low power consumption. Overall, the system presented herein is highly efficient and reliable and is therefore a strong candidate for deep-learning-based object detection on edge devices.

C. Execution on a General-Purpose Computer

The YOLOv3 model was also run on a general purpose computer using the Darknet framework. The experimental procedure involved downloading and compiling the Darknet API, loading the YOLOv3 pre-trained model, and executing the inference on the standardized test image.

Processing was performed on an AMD Ryzen 5 3500U CPU without GPU acceleration, providing a baseline comparison to the FPGA implementation. In this configuration, the computer required approximately 19 seconds to perform the detection task. While GPU acceleration could significantly reduce this time, it was omitted to ensure a fair comparison with the FPGA's dedicated hardware acceleration.

The PYNQ-Z2 FPGA significantly outperformed the CPU-based system, achieving a 23× reduction in inference time. This improvement highlights the advantages of using dedicated hardware for neural network inference, particularly in resource-constrained or real-time environments.

The YOLOv3 network deployed on the FPGA was a quantized version of the original model, optimized for the hardware's resource constraints. In contrast, the CPU-based implementation utilized the full-precision model. Quantization inherently reduces computational complexity but may result in a slight loss of accuracy. This trade-off underscores the importance of balancing performance and precision when deploying deep learning models on embedded systems.

TABLE III
PERFORMANCE METRICS COMPARISON OF YOLOV3 AND PYNQ-Z2 YOLOV3.

Platform	AP	AP50	AP75	APS	APM	APL
YOLOv3	33.00	57.90	34.40	18.30	35.40	41.90
PYNQ-Z2 YOLOv3	21.41	40.10	20.53	6.07	20.46	32.19

Although the PYNQ-Z2 demonstrated notable speed improvements, it is important to consider potential impacts on the accuracy of the model. Compression and quantization simplify the model parameters, which can affect performance metrics. However, the reduction in accuracy observed in this study was not proportional to the gain in speed, demonstrating the FPGA's efficiency in maintaining an acceptable balance between inference speed and precision.

V. Conclusion

In this paper, the target detection models YOLOV3, Tiny YOLO are deployed on the PYNQ Z2 board. Along with the standard data set, the real-time processing is also performed using same set up. The results obtained highlight the potential of FPGAs for deploying deep learning models in real-time applications. The PYNQ-Z2 offers a compelling combination of high-speed inference, energy efficiency, and adequate accuracy, making it a viable alternative to traditional computing platforms for embedded AI/ML applications. Additionally, its low power consumption further enhances its suitability for resource-constrained environments.

References

[1] Y. Wang, Y. Liao, J. Yang, H. Wang, Y. Zhao, C. Zhang, B. Xiao, F. Xu, Y. Gao, and M. X. et al., "An fpga-based online reconfigurable cnn edge computing device for object detection," *Microelectronics Journal*, vol. 137, p. 105805, 2023.

[2] D. Heller, M. Rizk, R. Douguet, A. Baghdadi, and J. P. Diguet, "Marine objects detection using deep learning on embedded edge devices," in *IEEE International Workshop on Rapid System Prototyping (RSP)*, Shanghai, China, 2022, pp. 1–7.

[3] A. Montgomerie-Corcoran, P. Toupas, Z. Yu, and C. S. Bouganis, "Satay: A streaming architecture toolflow for accelerating yolo models on fpga devices," in *International Conference on Field Programmable Technology (ICFPT)*, Yokohama, Japan, 2023, pp. 179–187.

[4] D.-D. Nguyen, D.-T. Nguyen, M.-T. Le, and Q.-C. Nguyen, "Fpga-soc implementation of yolov4 for flying-object detection," *Journal of Real-Time Image Processing*, vol. 21, no. 3, p. 63, 2024.

[5] Z. Valadanzoj, H. Daryanavard, and A. Harifi, "High-speed yolov4-tiny hardware accelerator for self-driving automotive," *The Journal of Supercomputing*, vol. 80, no. 5, pp. 6699–6724, 2024.

[6] R. A. Amin, M. Hasan, V. Wiese, and R. Obermaisser, "Fpga-based real-time object detection and classification system using yolo for edge computing," *IEEE Access*, vol. 12, pp. 73 268–73 278, 2024.

[7] A. Araujo, "Yolo on pynq-z2: Final results," gitBook publication, 2024.

[8] M. Dhouibi, A. K. B. Salem, and S. B. Saoud, "Cnn for object recognition implementation on fpga using pynq framework," in *2020 IEEE Eighth International Conference on Communications and Networking (ComNet)*, Hammamet, Tunisia, 2020, pp. 1–6.

[9] J. Q. et al., "Going deeper with embedded fpga platform for convolutional neural network," Beijing, China, 2020.

[10] Xilinx, *Zynq DPU Product Guide (PG338)*, 2024, available: https://www.xilinx.com.

[11] K. Sun, M. Koch, Z. Wang, S. Jovanovic, H. Rabah, and S. Simon, "An fpga-based residual recurrent neural network for real-time video super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 4, 2022.

[12] Xilinx, *DNNDK User Guide (UG1327)*, 2024, available: https://www.xilinx.com.

[13] V. H. Kim and K. K. Choi, "A reconfigurable cnn-based accelerator design for fast and energy-efficient object detection system on mobile fpga," *IEEE Access*, vol. 11, 2023.

[14] B. Gunay, S. B. Okcu, and H. S. Bilge, "Lpyolo: Low precision yolo for face detection on fpga," in *Proceedings of the 8th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'22)*, Prague, Czech Republic, 2022, pp. MVML 108–1, available: https://doi.org/10.11159/mvml22.108.

[15] S. Sun, J. Zou, Z. Zou, and S. Wang, *Experience of PYNQ: Tutorials for PYNQ-Z2*. Springer, available: https://doi.org/10.1007/978-981-19-9072-4.

[16] A. Lab, *FPGA Tutorial*, 2024, available: http://www.aiotlab.org/teaching/fpga.html.