FPGA Based ML Edge

Manish Sudumbrekar
Embedded and Cyber-Physical System
Irvine, USA
msudubr@uci.edu

Vishwanath Singh
Embedded and Cyber-Physical System
Irvine, USA
vishws1@uci.edu

Abstract—This report presents the FPGA-based ML Edge project, aimed at utilizing FPGAs to improve energy efficiency and performance in edge computing systems, specifically for ML-based vision algorithms. A camera is connected to an FPGA-based edge node to accelerate vision algorithms on images of lanes on road for lane detection. The project explores handling multiple accelerators on an FPGA, the allocation and scheduling of accelerator requests, and determining whether ML applications should run on the end device or the edge node. Key accomplishments include setting up the Xilinx Kria KV260 board, implementing lane detection using deep learning, and exploring various datasets. The report also covers short-term and long-term goals, including developing a real-time system for lane detection.

Keywords—FPGA, edge computing, machine learning, vision algorithms, energy efficiency, real-time processing, accelerator scheduling, Xilinx Kria KV260

I. INTRODUCTION

The FPGA-based ML Edge project aims to harness the power of Field Programmable Gate Arrays (FPGAs) to enhance the performance and energy efficiency of edge computing systems, particularly in the realm of machine learning (ML) and vision algorithms. Edge computing, which involves processing data closer to its source, is essential for applications requiring low latency and real-time processing, such as Advanced Driver Assistance Systems (ADAS) and autonomous driving. This project seeks to offload computationally intensive tasks from end devices to the FPGA, thereby optimizing resource utilization and reducing power consumption. The primary focus of the project is to implement a robust lane detection system using video footage captured by a camera mounted on the front windshield of a car. This system leverages the TuSimple and OpenLane datasets and employs the ENet-SAD model within the PyTorch framework to accurately detect road lanes. Through this project, we will gain insights into handling multiple accelerators on an FPGA, allocating resources efficiently, and managing the scheduling of ML tasks to achieve real-time processing capabilities.

II. OBJECTIVE OF THE PROJECT

A. Study and Analysis

Investigate FPGA design tools and ML engines for hardware accelerator generation. Analyse methods for transmitting images from end devices to the FPGA edge node. Implement ML-based vision accelerators on FPGAs. Develop policies for allocating multiple accelerators on an FPGA. Create scheduling policies to manage accelerator requests from end devices effectively.

B. Original Objectives for Spring Quarter

The original objectives for the Spring Quarter were focused on establishing the foundational setup and initial development for the FPGA-based ML Edge project. This included setting up the Xilinx Kria KV260 board and ensuring it was fully operational. Completing tutorials on the Vitis software environment was crucial to understand its functionalities and capabilities comprehensively. Additionally, we aimed to begin working with the OpenLane dataset to develop a lane detection model capable of navigating curved roads. To support ML model deployment, it was necessary to install the required SDKs and software on the Xilinx board.

c. Objectives Achieved for Spring Quarter

During the Spring Quarter, we successfully ran the ENet-SAD model on both the HP Zbook Firefly 15 G8 and the Acer Nitro 5 laptops, gathering important performance metrics for comparison. Additionally, we utilized the Vitis-AI library and Model Inspector to evaluate the feasibility of deploying the model on the Xilinx Kria KV260 FPGA in the future. This analysis provided valuable insights into the model's compatibility and performance on FPGA hardware, setting a strong foundation for subsequent development phases.

D. Updated Objectives for Fall Quarter Based on Progress

The updated objectives for the Fall Quarter are aimed at advancing the development and optimization of the FPGA-based ML Edge project. These include developing a fully functional ML application for lane detection within the Vitis environment using finalized datasets. We aim to complete all checkpoints on Vitis to ensure the lane detection application runs seamlessly on the FPGA. The entire process flow, from data acquisition to processing and output, will be implemented on the Xilinx Kria KV260 board. Detailed comparisons between FPGA performance and multicore processors will be conducted, focusing on metrics such as power consumption, latency, and throughput. Additionally, we will finalize the deep learning model architecture, including the number of layers and types of accelerators, to achieve optimal performance on the FPGA. Finally, we will work towards developing a real-time lane detection system capable of processing video feeds with minimal latency for demonstration purposes in December.

III. Setup Details for Project

A. Software Packages Used

1. Vitis Software: Version 2024.1, utilized for deploying and managing ML models and FPGA configurations.
2. Vitis-AI Library: Employed for running ML inference on the FPGA with efficient resource utilization.
3. PyTorch Framework: Used for developing and training the ENet-SAD model for lane detection.

TensorFlow: (version X.X) for additional deep learning experiments and model comparisons.

B. Hardware Used

1. Xilinx Kria KV260 FPGA: A high-performance, low-power FPGA platform designed for edge computing applications. It features a quad-core ARM Cortex-A53 processor and a programmable FPGA fabric.
2. Dashcam: Used to capture video footage of the road for lane detection.
3. HP Zbook Firefly 15 G8: With 32 GB RAM, Intel Core i7 processor, 65W power consumption, and 4400 mAh battery, used for initial software development and testing.
4. Acer Nitro 5: Featuring 32 GB RAM, AMD Ryzen 9 processor, 180W power consumption, and 3815 mAh battery, used for comparative performance testing.

Configurations	Hp Zbook Firefly 15 G8	Acer Nitro 5	Xilinx Kria KV260 FPGA
RAM	32 Gb	32Gb	4 GB
Power	65W	180W	36 W
Battery	4400 mAh	3815 mAh	7000 mAh
Processor	Intel core i7	Amd Ryzen 9	Xilinx Zynq UltraScale+ MPSoC with a quad-core ARM Cortex-A53 processor and a programmable FPGA fabric
Cost	2169$	1129$	249$

Figure 1: Summary of key hardware configurations used in the project

IV. Standards Used/Considered

A. Revision Control

To manage revisions and changes efficiently, we are using Git, a widely-adopted version control system. Git allows for robust tracking of code modifications, collaboration among team members, and maintenance of multiple versions of the project. This ensures that all changes are documented, and previous versions can be accessed or reverted to if necessary. By utilizing platforms such as GitHub or GitLab, we facilitate seamless collaboration and continuous integration practices, which are essential for the project's success.

B. Communication Protocols

For communication between the FPGA and other boards, we use both USB bus protocol and Ethernet. The USB bus protocol is employed to connect the FPGA to the camera, ensuring reliable data transfer for capturing real-time video footage. Ethernet is used for any additional communication needs between the FPGA and other boards, providing a robust and high-speed connection. These protocols are chosen to ensure efficient and reliable data transmission, critical for the real-time processing requirements of the lane detection system.

These standards and protocols play a crucial role in maintaining the integrity and efficiency of the project, enabling effective collaboration and real-time processing capabilities.

V. Iv. Prototypes

A. Initial Setup

Figure 2: Initial System Architecture Sketch

This initial sketch represents the basic layout of the system, illustrating how the Raspberry Pis are wirelessly connected to the FPGA edge node. The camera, mounted on the car's windshield, captures video footage of the road. This setup forms the foundation for developing the lane detection application.

B. Intermediate Prototype

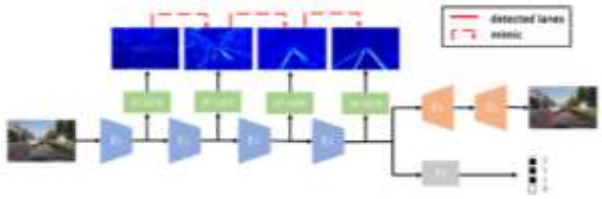

Figure 3: Block Diagram of Lane Detection System

The block diagram provides a detailed view of the data flow within the lane detection system. It highlights the process from image capture by the camera to data transmission to the FPGA, where ML-based lane detection algorithms are

executed. This diagram also shows the communication protocols used between the devices.

C. Software Setup and Testing

Figure 4: Screenshot of working lane detection

This screenshot shows the Vitis software environment where the lane detection ML models are developed and managed. It includes the interface for setting up the FPGA configurations and running ML inferences, essential for optimizing the system's performance.

D. Comparative Performance Testing

Performance Metrics	HP Zbook Firefly 15 G8	Acer Nitro 5
Performance	8/10	9/10
Power Consumption	65W	180W
Resource Utilization	75% average CPU usage	80% average CPU usage
Cost	$2169	$1129
Parallelism	16 threads	24 threads
Concurrency	8/10	9/10
Development Time	12 Hours	10.5 Hours
Scalability	7/10	8/10

Figure 5: Comparative Performance Chart

The performance chart illustrates the comparative analysis of the 2 multicore processors in terms of power consumption, latency, and throughput. This analysis helps in understanding the advantages of using FPGAs for real-time lane detection applications in edge computing environments in the spring quarter.

VI. Experiments

A. Test Setup

The current test setup involves running the machine learning model ENet-SAD on the TuSimple dataset to evaluate various performance parameters. The tests are conducted on two multicore systems (HP Zbook Firefly 15 G8 and Acer Nitro 5) and the Xilinx Kria KV260 FPGA using Vitis-AI Model Inspector. The primary objective is to compare the performance metrics between multicore processors and the FPGA.

B. Objectives of Test Cases

Measurement of Latency and Throughput for Training One Epoch: Objective: To assess the time taken to train one epoch of the ML model and evaluate the throughput of the system.

Measurement of Single-Core and Multi-Core Processing Times and Speedup Calculation: Objective: To compare the processing times between single-core and multi-core setups and calculate the speedup achieved through parallel processing.

Power Consumption Measurement for Both Linux and Windows: Objective: To measure and compare the power consumption of the systems running on Linux and Windows operating systems.

Monitoring of CPU Utilization During Training: Objective: To monitor and analyze the CPU utilization during the training process.

Threading for Parallel Processing and CPU Utilization Measurement After Multithreading: Objective: To implement threading for parallel processing and measure the CPU utilization to determine the efficiency of multithreading.

Figure 6: Screenshot of lane detection on curved roads

Figure 7: Screenshot of lane detection on curved roads

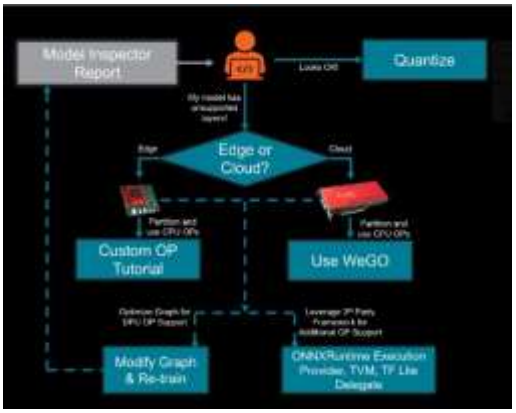

Figure 8: Working of Model Inspector

ACKNOWLEDGMENT

We would like to express our deepest gratitude to our mentor, Prof. Eli Bozorgzadeh, for his invaluable guidance and support throughout this project. We also thank Prof. Q V Dang for his assistance and constructive feedback. Their expertise and encouragement were crucial to our success. Thank you to all faculty and staff who provided resources and support.

REFERENCES

[1] M. SRIDHARA, "TuSimple," Kaggel, 2021. [Online]. https://www.kaggle.com/datasets/manideep1108/tusimple/code.

[2] AMD, "Vitis-AI," Github, [Online]. Available: https://github.com/Xilinx/Vitis-AI.

[3] AMD, "Vitis-Ai model inspector," AMD, [Online]. https://github.com/Xilinx/Vitis-AI/blob/v3.5/src/vai_quantizer/vai_q_pytorch/example/jupyter_notebook/inspector/inspector_tutorial.ipynb