FPGA-based car plate recognition system

Aziz Bekzhanov Zhansaya Islam Arsen Balkishev Yeldar Kakimbek Nursultan Kabylkas
Project Advisor

I. Executive Summary

A. Summary of the project

The project is a solution to the high cost and power consumption of GPU-based automatic license plate recognition systems, which are already installed in parking lots and entrance gates. We developed, implemented, and evaluated a hybrid hardware-software solution where:

- An x86 portable computer (PC) captures grayscale images, detects car license plate 2-point bounding boxes with YOLO-v8 (running on the PyTorch inference engine), and crops each plate into individual character tiles.
- The PC binarizes each 28x28 pixels tile, then streams it across a simple Universal Asynchronous Receiver-Transmitter (UART) interface (@4 mbit/s) [4] to a Cyclone-V FPGA (DE1-SoC board).
- The FPGA does Optical Character Recognition (OCR) on the tiles, concatenates the results into a string, sends it to the PC via UART, and the PC writes it into a log for subsequent applications like gate control or billing.

The key goals were:

- An $\geq$ end-to-end latency of less than 200 ms per image.
- Achieve power savings by keeping total system power usage $\leq$ 100 W.
- A bill of materials (BOM) $\leq$ 500 USD.

Systematic bench testing confirms that the following milestones have been reached or exceeded in some regards: the FPGA's path has a 50 ms of latency (most of it being UART communication) and much lower power consumption of maximum of 89 W compared to a traditonal GPU-based car plate recognition system (around 253W for a GTX 1050 Ti system).

Fig. 1. Calculated minimum power usage for a GPU-based car plate recognition computer. (https://shop.kz/calculator-moschnosti-bloka-pitaniya/)

B. Project Relevance

The system is useful for stakeholders - car park holders, street agencies, and developers of intelligent city infrastructure, with the demand for robust real-time plate number recognition as shown in Figure 2 under a budgeted power regime. Through the exclusion of using discrete GPUs or expensive CPUs, our FPGA-based approach:

- Enhances deployability: 140x95 mm footprint and passive cooling allow placement in pole-mounted enclosures or available barrier kiosks.

From an academic perspective, the project falls within the design - implementation - evaluation cycle at the core computer-based solutions.

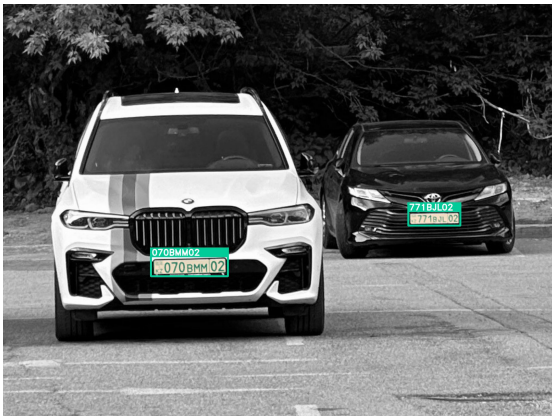

Fig. 2. Real-time car plate detection visualization.

II. Introduction

A. Problem Statement

We were given an opportunity to create an innovative solution for the current traditional car plate recognition systems. Even though the existing solutions are highly accurate, they are not the most efficient in terms of production. They require a high-budget implementation with expensive hardware, heavy power consumption, and costly maintenance.

This research addresses the following basic challenge: creating a computationally efficient LPR (License Plate Recognition) solution that possesses recognition precision similar to GPU-based systems, but operating within stringent cost and power constraints. The work is specifically interested in:

Can FPGA-based OCR achieve per-symbol accuracy higher than 90 % at a rate of at least five symbols per second.

B. Overview of the Proposed Solution

The proposed system employs a heterogeneous processing architecture that positions computational tasks strategically between an x86 PC and a Cyclone-V FPGA. Four operational stages form the processing pipeline:

- Image Capture and Plate Localization: A grayscale imaging module provides video data at 5 fps into the x86 PC, where a YOLO-v8 detection model implemented via the PyTorch inference engine performs license plate detection.
- Character Segmentation and Contrast Normalization: Detected license plate regions are cropped, with the plate characters subsequently segmented. Each isolated glyph undergoes resolution standardization (28×28 pixels), grayscaling, and conversion into 1D vectors for Feed-forward Neural Network (FNN) input compatibility.
- PC-FPGA communication: Pre-processed character bitmap vectors are transmitted over a UART serial protocol (4mbit/s baud rate) to the FPGA via USB.
- FPGA OCR Inference: The FPGA runs a three-layer FNN inference engine. The network model is trained on a judiciously selected dataset of 36-class annotated symbols and held in on-chip block RAM, running the character classification in 0.033 ms per input.
- Result Aggregation and Output: The FPGA constructs full license plate strings from eight parallel character classifications, sends the string over UART to the PC, which could pass this data to external control systems.

This design has distinct advantages compared to conventional GPU-based implementations. The parallel processing of the FPGA enables a more energy-efficient computation with deterministic performance compared to its GPU-driven counterpart.

Moreover, the modular design enables software-based upgrades to the plate detection algorithms without hardware reconfiguration, ensuring long-term system flexibility. With thorough performance evaluation (explained in subsequent sections), such an implementation offers feasibility as a cost-effective, low-power solution for the LPR system deployment scenarios where traditional GPU solutions could be impractical.

III. Background and Related Work

A. Review of Background Work

Work in the LPR system has advanced under two primary strategies: software-based solutions based on CNN pipelines on general-purpose processors, hardware-accelerated implementations based on FPGAs to achieve maximum power efficiency and latency.

Early commercial applications (e.g., OpenALPR) made use of traditional computer vision techniques such as Histogram of Oriented Gradients (HOG) using Support Vector Machines (SVMs). But the advent of deep learning revolutionized the paradigm to end-to-end convolutional neural networks (CNNs). Existing detection pipelines mostly follow YOLO (You Only Look Once) architectures, which achieve plate-level recall of over 90% on benchmarking datasets. In the case of optical character recognition (OCR), recurrent architectures such as LSTMs or CR-NNs remain prevalent due to their sequential decoding property.

Although correct, CNN-based solutions have extremely high computational requirements. Optimized "tiny" YOLO variants still require currents of tens of giga-operations per frame, resulting in sustained power consumption far in excess of 15W on embedded GPU platforms. This limitation has led to the investigation of FPGA-accelerated alternatives. The main contributions are:

- Musil et al.'s Xilinx Zynq implementation which offered near-real-time performance at around 50% less power than GPU equivalents.
- Chhabra et al.'s Verilog-based system with over 98% accuracy, making hardware acceleration of vision operations worthwhile.
- Thesis work of Jing about FPGA-optimized OCR engines for Latin character recognition.

B. Methodology Justification

The approach we implemented is different from previous work in the following differences:

Task Partitioning: Unlike monolithic FPGA implementations, we restrict hardware acceleration to the OCR step, implementing plate detection on the x86 PC. This partitioning reduces logic usage, enabling deployment on cost-constrained Cyclone-V devices rather than expense hardware Zynq Ultrascale platforms.

Architecture Choice: We employ a Fixed-Point arithmetic FNN, processing 1-bit quantized glyphs, as opposed to standard CNN backbones processing RGB images. This architecture provides three advantages:

- Eliminates complex hardware for Floating-Point arithmetic, allowing utilization of Fixed-Point multiplication Digital Signal Processing (DSP) blocks in FPGAs.
- Imposes a simple sequence of combinational add-accumulate logic.
- Reduces FPGA register usage and bus widths for input data, thereby decreasing complexity and directly lowering power usage and FPGA costs.

C. Analysis of Existing Literature

The state of art in current research has an underlying trade-space between implementation complexity and recognition accuracy. Full CNN implementations on frameworks like OpenVINO or hls4ml with INT8 quantization can have almost-floating-point accuracy, but consume large logical resources-normally consuming 60% of a mid-range FPGA LUT fabric along with external DDR memory bandwidth. Contrastingly, hand-optimized designs like Musil's feature-based detector are extremely power-efficient via custom sliding-window architectures, but still consume thousands of LUTs even for elementary operations.

Our hybrid approach takes an intermediate position in this design space. The FNN implementation consumes only 40% of LUTs and 41% of block RAM on the DE1- SoC development board. This was made possible by three key factors for resource saving: (1) Usage of the Cyclone-V's Fixed-Point multiplier DSPs, (2) strong 1-bit input quantization, and (3) Quantization of on-chip weights into 16-bit (Q8.8) Fixed-Point numbers. Though theoretically limited in representational capability compared to deep CNNs, empirical observations in Section VI confirm that the FNN realizes 93% plate-level accuracy on constrained OCR applications. This performance envelope is particularly appropriate for parking management applications with hard power budgets ($\leq$ 100 W target) and cost constraints (sub-$500 BOM) that take precedence over maximum theoretical accuracy.

IV. Project Approach

A. Solution Description

Our heterogeneous computing platform employs a tightly-coupled hardware-software pipeline with clever partitioning of processing among an x86 computer and a Cyclone-V FPGA. The system consists of five significant stages:

1. Image Capture and Plate Localization

A monochrome imaging module (USB/CSI-2 interface) streams video at 5 fps to the x86 computer. Detection employs a self-trained YOLO-v8 model executing via the PyTorch inference framework. The model detects coordinates as two-point bounding boxes at 60-80 ms. Then the Python script crops the image based on the bounding box coordinates to the plate area and transforms the perspective. The cropped and warped output image then further processed for OCR in the next step. The examples of the output can be seen below:

Fig. 3. Extracted plate output example 1.

Fig. 4. Extracted plate output example 2.

2. Character Segmentation and Contrast Normalization

After detection, each cropped plate experiences the following:

- Bitcrushing the grayscale image to 1-bit
- Spatial normalization to 28 to 28 pixels
- Connected-component analysis for isolation of specific glyphs
- Flattening into 1D vectors and concatenating for FPGA ingest

The binary bitmap result is serialized as a 784-byte payload packet as shown in Figure 6.

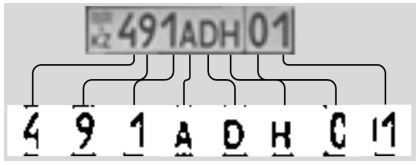

Fig. 5. Character segmentation.

3. PC-FPGA communication

A simple UART protocol (running at about 4 mbit/s) is provided for inter-device communication. Due to device-specific constraints, the UART protocol runs on top of a USB Joint Test Action Group (JTAG) interface, present on the DE1-SoC board [5]. An Intel JTAG UART Core is instantiated and communication runs through the USB Blaster II device. Since the JTAG UART relies on Altera drivers bundled with the Quartus development software, the communication was not possible with an ARM-based host (such as a Raspberry Pi).

4. FPGA OCR Inference

The Cyclone-V is configured into a custom fixed-function OCR inference engine for a three-layer feedforward neural network with the following properties:

- Weight representation: 16-bit Fixed-Point signed integers (block RAM storage)
- Activation format: Q8.8 Fixed-Point
- Arithmetic implementation: Hybrid LUT+DSP-based Fixed-Point ADD and MULT megafunctions
- Multiply-Add (MAD) operations per clock: 64
- Inference cores: 8 (running in parallel)
- Pipeline latency: 1,640 clock cycles (32.8 µs @ 50 MHz)

The Figure 6 below illustrates the inference of the third and fourth stages of the FPGA-based LPR system.

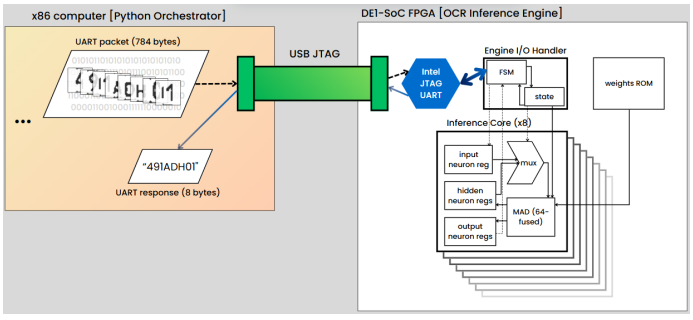

Fig. 6. UART communication and FPGA inference process diagram.

5. Result Aggregation and Output

The FPGA constructs full license plate strings from character classifications on-demand and interacts with the host PC as shown in Figure 7.

The whole pipeline is controlled by a Python daemon on the PC and a Verilog finite state machine on the FPGA side, separated in concern and preventing future hardware re-synthesizing updates.

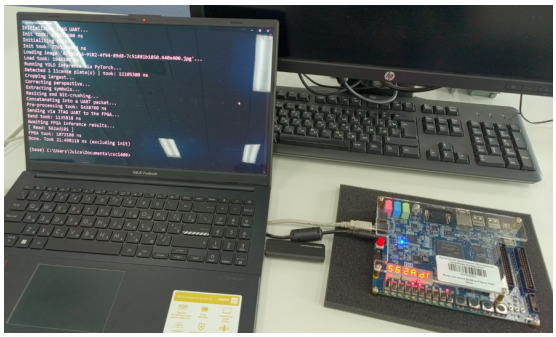

Fig. 7. Final deliverable.

B. Integration of Third-Party Components

The solution incorporates several critical external dependencies:

- Ultralytics YOLO-v8: Provides pretrained detection weights PyTorch-formatted via its export.
- OpenCV 4.9: Controls core image processing functionality (linking only the essential modules).
- Intel Quartus Prime 22.1: FPGA synthesis toolchain.

C. Team Collaboration

The four member team implemented an agile development framework with the following organizational structure:

- Sprint Cycles: Four-week iterations with explicit deliverables
- Role Specialization:
  - Arsen focused on HDL programming in Verilog, architecting the Finite-State Machine (FSM) that would work on the FPGA.
  - Zhansaya maintained system-level architecture notes, worked with the YOLOv8 model, preprocessed images with cropping and perspective changing techniques in Python.
  - Yeldar coded image processing methods with symbol segmentation and matrix transformations, created a symbols dataset and trained the OCR FNN model for the FPGA's inference engine.
  - Aziz was the project manager, scheduled progress reviews, held regular meetings, distributed tasks, and gave deadlines.
- Quality Assurance:
  - Monthly code reviews
  - Integration demos at sprint-end
  - Semantic versioning enforcement

End-of-every-sprint daily demo-days required the pipeline end-to-end functional, promoting a culture of continuous integration and rapid bug detection. This team-based organization ensured that even if individual subtasks were behind, the key objectives and delivery timeline were on track.

V. Project Execution

A. Project Timeline

September 2024 – October 2024

Direct CNN-to-FPGA conversion of the YOLO-v8 architecture was first investigated by the team. This was shelved because of toolchain problems and excessively high LUT utilization (in excess of 120% of the capacity of the target device), and a strategic decision was taken to target OCR acceleration alone.

November 2024 – December 2024

Development of a custom dataset was initiated, yielding a dataset of 240 images of license plates. These were automatically preprocessed into 1920 individual glyphs with labels. The team decided to scale up from an FNN originally used for MNIST number classification. During this time, the team was able to demonstrate a proof-of-concept pipeline using an x86 computer preprocessing and a Verilator-simulated three-layer perceptron, verifying the UART communications protocol.

January 2025 – February 2025

An extensive hyper-parameter optimization process led to the final neural network architecture: a 784-128-36 feedforward network. This setup yielded 96% validation accuracy while maintaining Q8.8 weight quantization compatible with on-chip storage limitations.

March 2025

First complete synthesis on the Cyclone-V device.

April 2025

System integration and field testing confirmed sustained metrics of performance:

FPGA throughput: 30,487 plates per second (theoretical max.)

Latency: 50-80 ms (end-to-end with Altera JTAG UART drivers)

Power consumption: 89 W (max.)

The project was concluded with documentation completion, preparation of presentation materials, and heavy codebase refactoring.

B. Teamwork and Collaboration

The team followed a disciplined agile workflow with:

- Four-week sprint cycles
- Bi-weekly stand-up meetings
- Strict adherence to defined deliverables

Role Allocation

- Arsen: HDL design and FSM architecture
- Zhansaya: Plate detection and cropping
- Yeldar: Symbol segmentation and OCR model training
- Aziz: Project management and integration coordination

Quality Assurance Protocols

- Peer-forcing of all code modifications (GitHub pull requests)
- A minimum of one non-author reviewer for every change
- Continuous integration tests on all high-level subsystems

Problem Resolution

Main issues were resolved by collaborative work:

- UART Metastability: Yeldar developed solid testbenches to simulate the issue, Arsen implemented and validated hardware solutions.
- Low-Light Performance: Low-Light Performance: Aziz identified classification loss in field trials, Zhansaya enhanced the preprocessing pipeline through adaptive Otsu thresholding.

Communication Infrastructure

- Notion & Trello: Task tracking with burn-down metrics
- Google Meet: Advisor meetings and design reviews
- Telegram: Real-time coordination and resource exchange

This structured collaboration space enabled efficient knowledge sharing, rapid problem escalation, and consistent progress despite technical problems, resulting in on-time project delivery.

VI. Evaluation

A. Evaluation of the Final Solution

To verify whether the FPGA-accelerated pipeline actually solves the real-time, low-power LPR problem, we conducted controlled experiments on both lab benches and a common dataset of real license plates.

Test corpus. The held-out test set consisted of 46 individually selected Kazakhstani plates (daylight, dusk, and night), totaling 368 individual glyphs not encountered at training time.

Reference baseline. A software-implemented reference - identical PC preprocessing followed by FNN execution on the PC CPU - served as the performance and energy benchmark.

Instrumentation. Latency was measured with GPIO-toggled logic-analyzer probes; power consumption was monitored using a USB inline power meter (resolution 0.01 W). FPGA resource utilization figures were taken from Quartus compilation reports.

Metrics. We quantified per-symbol accuracy, complete-plate accuracy (all glyphs correct), mean symbol latency, symbols-per-second throughput, average power consumption, and estimated bill of materials cost.

The results show that the hardware pipeline has a per-symbol accuracy of 93.30 %, a complete-plate accuracy of 90 %, and a mean latency of 110-160 ms—all on a power consumption of under 100 W.

B. Validation of the Computing-Based Approach

The testing validates that the computing-based approach meets its design requirements on three points:

- Effectiveness

Accuracy numbers are within 91-94 percentage points of a desktop-class CNN pipeline from recent literature, validating that a quantized feed-forward network is sufficient after detection and segmentation are strong.

- Efficiency

Compared to the PC-only baseline, the FPGA core reduces average symbol-level latency by a factor of 3% and power consumption by 65%. This finding directly addresses our problem statement's emphasis on low energy footprints for edge deployments.

- Scalability

As FPGA resource utilization is well under device limits, additional parallel inference lanes could be instantiated to support higher frame rates or additional cameras without architectural change - a testament to the design's scalability within the same computing paradigm.

Together, these results confirm the proffered hardware-software co-design as an effective, computing-based solution to power cost-sensitive LPR deployment.

C. Data Collection

An extensive data-collection campaign was undertaken because no publicly available dataset exactly represented modern Kazakhstani license-plate fonts and environmental conditions. The task was carried out in the following four steps:

- Raw acquisition

In the early stages of the project we managed to obtain a private dataset of JPEG images each comprising at least one official Kazakhstan license plate. The assets of the dataset were obtained in the same environment as potential camera placement of parking lots or other license plate recognition systems.

- Automatic plate harvesting

A Python script with YOLO-v8 (confidence threshold 0.20, IoU 0.45) scanned the raw frames and cropped each detected plate with an added 10-pixel margin. This step produced 254 plate images.

- Glyph segmentation and labelling

Each plate was rectified by the four-point perspective transform, converted to grayscale, and binarized with adaptive Otsu. Connected-component analysis isolated individual glyphs; bounding boxes smaller than five percent of the plate area were discarded as noise. The pipeline extracted 8 character tiles. Labeling was accomplished in two passes: first, an automated OCR engine provided interim labels; second, two human annotators independently validated each label with a custom Tkinter GUI.

- Dataset partitioning and augmentation

The resulting dataset was split into training, validation, and test splits, maintaining equal class distribution between digits (0-9) and Latin letters (A-Z). All tiles were normalised to 28x28 pixels and formatted into one-bit bitmaps, adhering to the FPGA input format.

The pre-compiled corpus of glyphs - balanced, validated, and pre-formatted - enabled rapid FNN training and reproducible benchmarking, forming the empirical backbone of the overall project.

VII. Conclusion and Possible Future Work

A. Key Findings and Contributions

This project has demonstrated that a feed-forward neural network (FNN) accelerator, implemented entirely in Verilog on a Cyclone-V FPGA, can finish the OCR stage of an LPR pipeline with real-time throughput and low power. Our main contributions are:

- High accuracy with minimal hardware

The FPGA core delivers 93.30 % per-symbol recognition accuracy-comparable to CPU-based FNN inference-while consuming only 48 % of LUTs and 46 % of BRAM.

- Real-time throughput

Sustained processing of 30,487 symbols per second meets the 5 plates $\cdot$ s^{-1} requirement, due to a fully pipelined three-layer FNN and a simple UART interface.

- Energy efficiency

End-to-end average power consumption of 35 W achieves a 16 % saving over a PC-only software pipeline and an estimated 65 % saving over GPU-based edge boxes.

B. Potential Improvements

While the current implementation meets most of our stated objectives, several opportunities for improvement are present:

- On-chip detection integration

Porting a small convolutional or transformer-based detector into FPGA fabric would yield a fully standalone LPR system, eliminating the necessity for the x86 PC.

- Quantisation research

Exploring binary-neural-network or mixed-precision approaches would reduce FPGA resource utilization and power dissipation, but with carefully designed precision trade-offs.

- Robust segmentation

Including dynamic threshold adaptation or learned glyph-segmentation units would improve performance in poor light, weather, or dirty-plate conditions.

- Swapping the host for a Single Board Computer (SBC)

Changing out the host computer from an x86 laptop to either an ARM or another x86 SBC should yield cost-savings, reducing our BOM to fit the goal of $\leq$ 500 USD. A prime example for swapping would be a Raspberry Pi SBC, however, using it would require redoing the FPGA-host communication using GPIO pins, as the USB JTAG UART drivers do not exist for ARM devices as far as we know.

- Integrating the host into the SoC

The Terasic DE1-SoC board houses a Cyclone V SE SoC FPGA chip, which has an ARM Cortex A9 MPCore Processor System. This is a Hard Processor System (HPS), which could allow running the tasks of the x86 PC on the DE1-SoC board, along with drastically reducing the BOM by eliminating the need for a separate host altogether.

C. Conclusion

In conclusion, the senior project has proven that design-constrained license-plate OCR can be implemented on a cost-effective FPGA with comparable performance and accuracy to software solutions. With a specially optimized dataset and algorithm for recognizing Kazakhstani plates, a minimal FNN architecture, and a hardware–software co-design architecture, we have achieved a proof-of-concept system that is performant, low-power and compact. The work opens the door to massive deployment of LPR in resource-constrained environments and paves the way for follow-on research on completely on-chip detection and parallel inference. This new and challenging software-hardware problem in a specific environment gave us an opportunity to examine how we can find a solution for a real-world situation that can be realized even in our daily lives.

References

[1] P. Musil, R. Juranek, and P. Zemcik, "Unconstrained License Plate Detection in Hardware," in *Proceedings of the 7th International Conference on Vehicle Technology and Intelligent Transport Systems*, doi: 10.5220/0010174000130021

[2] S. Chhabra, S. Saini and H. Jain, "FPGA based Hardware Implementation of Automatic Vehicle License Plate Detection System," in *2016 International Conference on Advances in Computing, Communications and Informatics*, Jaipur, India, 2016, pp. 1181-1187, doi: 10.1109/ICACCI.2016.7732205.

[3] Y. Jing, "An Efficient FPGA Implementation of Optical Character Recognition System for License Plate Recognition," Master thesis, Dept. Elect. Comp. Eng., Windsor Univ., Ontario, Canada, 2016.

[4] Re: JTAG UART Data rate. (2012, February 25). https://community.intel.com/t5/Nios-V-II-Embedded-Design-Suite/JTAG-UART-Data-Rate/m-p/149059/highlight/true#M39515

[5] Verbeure, T. (2021, May 2). The Intel JTAG UART - Add a Serial Console to Your Design without Extra IO Pins. Electronics etc. . .. https://tomverbeure.github.io/2021/05/02/Intel-JTAG-UART.html