Architectural Design Exploration of a Lane Detection Vision Pipeline for FPGA-based F1Tenth Autonomous Vehicles

Andrea Magnani
University of Modena and Reggio Emilia
Modena, Italy
229952@studenti.unimore.it

Gianluca Brilli
University of Modena and Reggio Emilia
Modena, Italy
gianluca.brilli@unimore.it

Andrea Marongiu
University of Modena and Reggio Emilia
Modena, Italy
andrea.marongiu@unimore.it

Abstract

Heterogeneous System on Chip (HeSoC) based on reconfigurable accelerators, such as Field-Programmable Gate Arrays (FPGAs), offer a promising solution to meet the performance and energy efficiency demands of advanced perception and localization tasks in autonomous vehicles. This study investigates the hardware acceleration of a computer vision pipeline for lane detection on an FPGA, specifically targeting the AMD Kria KV260 device. We evaluate various integration architectures, memory organizations, and offloading strategies for integrating the multiple components of the pipeline. Experimental results demonstrate that the proposed solutions achieve up to 22× speedup compared to a software-only implementation, highlighting significant improvements in resource usage and processing efficiency.

CCS Concepts

• Hardware → Hardware accelerators; **• Computer systems organization → System on a chip**; **Robotic autonomy**; **• Computing methodologies → Computer vision**.

Keywords

Heterogeneous System on Chip, FPGA, Autonomous Vehicles, Lane Detection, Computer Vision, AMD Kria KV260

ACM Reference Format:
Andrea Magnani, Gianluca Brilli, and Andrea Marongiu. 2025. Architectural Design Exploration of a Lane Detection Vision Pipeline for FPGA-based F1Tenth Autonomous Vehicles. In *22nd ACM International Conference on Computing Frontiers (CF Companion '25), May 28–30, 2025, Cagliari, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3706594.3726973

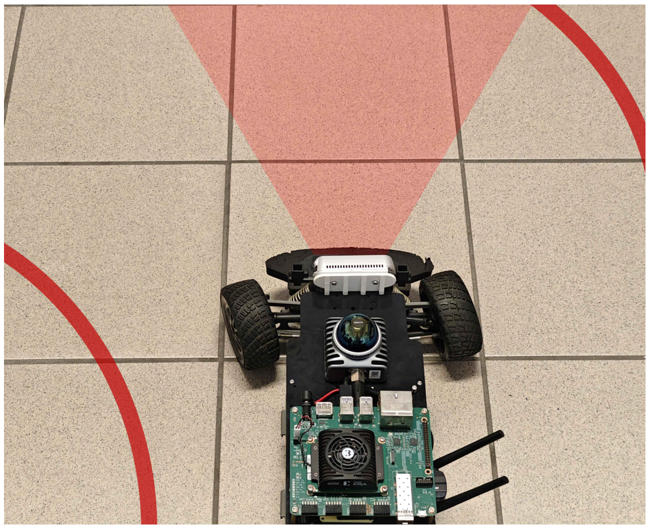
Figure 1: Example of a lane detection system for *F1Tenth* autonomous cars.

1 Introduction and motivation

Lane detection is the process of identifying lane markings on a road using image processing techniques. It plays a crucial role in the field of autonomous driving, ensuring that autonomous cars follow the correct path. Traditional lane detection methods rely on techniques such as edge detection, color filtering, and the Hough Transform to recognize lanes. These methods analyze contrasts and geometric patterns to distinguish lane markings from the rest of the road. More advanced approaches use curve fitting and perspective transformation to improve accuracy in detecting lane structures [8]. Luo et al. use a neural network and deep learning-based approach, based on a transformer architecture, instead of relying on traditional computer vision techniques [9]. Hybrid approaches try to combine neural networks with computer vision techniques [12]. Lane detection operates in real-time, processing continuous frames from a camera mounted on the vehicle. To achieve the necessary speed and efficiency, these algorithms are typically implemented on small and constrained embedded systems, utilizing hardware accelerators such as GPUs, FPGAs, or dedicated processors. This ensures fast computation while meeting the constraints of automotive applications. A relevant use-case of lane detection is the *F1Tenth* competition[1], a platform for autonomous racing with 1:10 scale cars. Due to their small size, these vehicles are equipped with compact and low-power computing platforms, which impose constraints on processing capabilities. To achieve real-time performance, efficient implementation of algorithms is essential. FPGAs are widely adopted to implement lane detectors due to their ability to process data in parallel while maintaining low power consumption. However, programming FPGAs using traditional hardware description languages (HDLs) can be complex and time-consuming, and requires specialized expertise. To streamline development, on one hand, High-Level Synthesis (HLS) tools enable developers to

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CF Companion '25, Cagliari, Italy
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1393-4/25/05
https://doi.org/10.1145/3706594.3726973

[1]https://roboracer.ai/

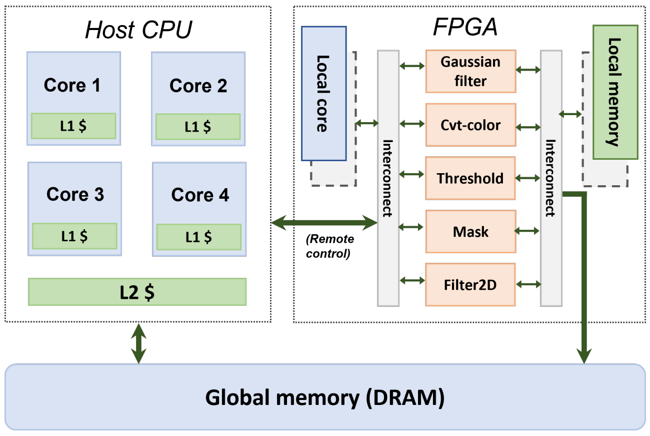

Figure 2: FPGA-based HeSoC with the five accelerated kernels using AMD Vitis Vision.

write FPGA-accelerated code using higher-level languages such as C/C++ [7], while on the other hand, overlay architectures try to simplify the deployment of accelerator-rich architectures [5, 6]. Furthermore, many FPGA vendors and academia, provide optimized libraries or IPs, to simplify implementation [1, 3, 4]. For example, Vitis Vision offers a collection of pre-built and highly optimized image processing functions, implemented in HLS and tailored for FPGA acceleration [3]. Several works use Vitis Vision to accelerate computer vision pipelines on AMD FPGAs [10, 11]. Nakagawa et al. use Vitis Vision library to accelerate a computer vision system for height measurements on mono camera-equipped drones [10].

The contribution of our paper is the acceleration of part of the lane detection application on FPGA. This system serves as a key building block for autonomous driving on the *F1Tenth* platform, allowing us to test and evaluate various architectural optimizations. All the code and design associated with this project are released in an open source repository[2].

2 Design Exploration

Our reference system is based on a heterogeneous FPGA architecture, where a multicore CPU (ARM Cortex A53) is coupled with programmable logic, forming a unified computational platform. The main memory is shared, enabling data exchange between CPU and FPGA. The starting point of our work is the software implementation of the entire lane detection pipeline on the ARM Cortex A53[3], which also serves as the reference point for our performance evaluation. With our experimental campaign we identified kernels to be accelerated on the FPGA. We deployed hardware accelerators using the AMD Vitis Vision library [3], a collection of libraries composed of some common computer vision functions. The Vitis Vision libraries include different utilization flows that are organized in three different levels, named *L1*, *L2* and *L3*. We used the L1 flow, that contains low-level HLS implementations, while the other two levels (L2 and L3), also include OpenCL code for interactions with the FPGA and more complex applications, composed of pipelined kernels. This approach allowed us to rapidly develop and integrate custom accelerators while benefiting from state-of-the-art library components.

As reported in Figure 2, all hardware accelerators are memory-mapped and managed by the host CPU, which orchestrates the overall execution flow. Each accelerator is equipped with master AXI interfaces for high-bandwidth data transfers, with all AXI links multiplexed onto a shared main interconnect. Moreover, we evaluate the impact of several architectural optimizations to further enhance performance and resource utilization. These optimizations include: i) the introduction of local memory buffers to minimize data transfer latency; ii) pipelined execution to increase throughput; iii) local control units to reduce communication overhead between the CPU and accelerators.

Global vs local memory. With this architectural optimization we aim to quantify the benefits achieved using private local memories for each stage of the computer vision pipeline, in contrast to the baseline architecture where all data exchanges relied on shared DRAM. In the proposed approach, the input data is initially read from the DRAM before being processed by the pipeline. During the intermediate stages, data exchanges between subsequent processing steps are handled entirely through local memory, which is dedicated to each computational unit. This strategy reduces the requests on the shared DRAM and minimizes latency due to memory access contention. Once the final stage of the pipeline is completed, the resulting output data is written back to DRAM for further use. This approach aims to exploit the lower access latency of private memories, improving the overall performance.

Remote vs local control. In this configuration, a local processor core — implemented as a softcore deployed on the FPGA fabric — is physically closer to the accelerator compared to the remote control scenario where the host processor manages the accelerators. The communication path between the softcore and the accelerator's register file is shorter, requiring fewer hardware interconnects to be traversed. This architectural optimization reduces the latency for reading and writing the accelerator's control registers, enabling more lightweight and frequent interactions. Moreover, parameter control is performed via an AXI-Stream interface, unlike the *Remote control* configuration where the AXI-Lite protocol was employed. This more efficient communication mechanism contributes to faster reconfiguration of the accelerator and overall improved system performance.

Sequential vs pipelined model. On the software side, we analyzed two different execution models to orchestrate the accelerator kernels. The first approach is the *sequential model*, where the kernels are executed on after the other in a strictly sequential manner. The second approach, reported in listing 1, is the *pipelined model*, which exploits parallelism to improve performance. In this model, all kernels are launched in parallel, using a double buffering memory scheme. The execution is organized in stages: during each stage, all accelerators process their respective data in parallel, and the next stage can only be launched once every accelerator in the current stage has completed its task. This model aims to overlap computation and data transfer, potentially reducing overall execution time by maximizing resource utilization.

[2]https://github.com/Magnani95/lane-detection-xilinx
[3]https://github.com/MichiMaestre/Lane-Detection-for-Autonomous-Cars

Listing 1: Pipelined execution model

```
/* 1. Define arrays of kernel functions to
    start and wait */
/* 2. Start all kernels */
for (int i = 0; i < 5; i++) {
    start_kernels[i]();
}
/* 3. Wait for kernels completion */
for (int i = 0; i < 5; i++) {
    wait_kernels[i]();
}
```

3 Experimental Results

The experimental platform used to validate our approach is the AMD Kria KV260 [2], a heterogeneous system-on-Chip (HeSoC) designed for edge computing and AI applications. This platform features a quad-core ARM Cortex-A53 processor for general-purpose applications and a dual-core ARM Cortex-R5 processor optimized for real-time tasks. Our focus is on implementing and optimizing a lane detection system, leveraging the FPGA resources provided by the KV260 platform.

To support the lane detection pipeline, we designed a custom architecture based on hardware accelerators, implemented via HLS. The accelerators are tailored for image pre-processing, feature extraction, and the geometrical analysis required for lane detection. We first conducted a software profiling of the kernels in the lane detection pipeline, reported in Table 1. This step allowed us to identify the computational bottlenecks and assess the performance-critical stages of the application. We then evaluated the execution times of the various kernels and we identified *gaussian filter* and *Hough transform* as the most compute-intensive kernels of the pipeline. However, due to the limited FPGA resources available on the Kria KV260, we were unable to accelerate the *Hough transform*, as its implementation exceeded the available area. Instead, we focused on accelerating the *gaussian filter* along with the subsequent four kernels in the pipeline (reported in bold in the table), while the rest of the kernels are executed in software.

3.1 Methodology

In our experiments, reported in Table 2, as described in the previous section, we integrated HLS kernels from the Vitis Vision on the FPGA. In the first group of experiments, referred as *Global memory & remote control*, all the input/output memory ports of the kernels are conneted to the global DRAM, while the host CPU (ARM Cortex A53) directly controls the kernel execution. As we can note the *gaussian filter*, that was the most compute-intensive stage, got the better speedup ≈ 92× compared to the software execution using OpenCV. However due to the last four kernels that are still implemented in software, the speedup of the lane detection is ≈ 2×. In the following subsections we analyze the impact of different system architectures on the end-to-end execution time of the lane detection application.

Exp 1 — Global vs local memory. The private memories are implemented in the Vivado block design using *Block Memory Generator* component and *BRAM* resources. Block RAMs are memories located on the FPGA that allow low latency and more predictable accesses. Our experiments are referred as *Local memory & remote control* in Table 2. As we can note from the table, due to different access patterns, not all the kernels benefit from the use of local memories. For example considering *Filter2D* the speedup increases from ≈ 7.5× up to ≈ 7.7×, while the *Mask* kernel increased its speedup from ≈ 2.6× up to ≈ 5.0×. ***Not all kernels benefit from local memories, since some are compute-bound, where their execution time is primarily determined by computation. Accelerators that are memory-bound gain the most from this optimization.***

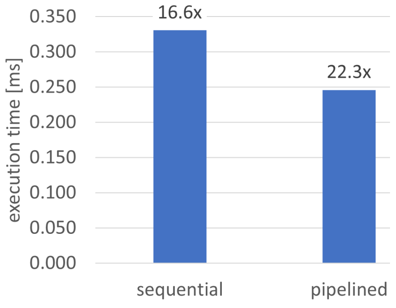

Figure 3: Execution time and speedup of *Sequential vs pipelined model.*

Exp 2 — Remote vs local control. The last column of Table 2 reports the performance achieved using a local control, referred as *Local memory & local control*. As we can note from the table, in this case we don't have a significant improvement in terms of achieved speedup. ***The kernels that exhibit more benefits from a local control, are those kernels that require a high number of parameters before their execution*** (i.e. a high number of CPU/accelerators interactions). In our case the *gaussian filter* and *Mask* are the kernels that are characterized by a greater speedup compared to the others. For example considering the *gaussian filter*, the speedup from ≈ 92× reaches ≈ 96×, while Mask from ≈ 5× to ≈ 5.6×. Considering the five accelerated kernels, we have an average speedup of 17.5× compared to the software execution. While if we consider the full lane detection pipeline (with more four kernels implemented using software OpenCV) we have a 2.1× of speedup, that translates into 217 frames processed per second.

Exp 3 — Sequential vs pipelined model. To evaluate the benefits of the pipelined execution model, we compared its performance against the sequential execution using the best-performing configuration from previous experiments (i.e. the last column of Table 2). As reported in Figure 3, the sequential model achieves a speedup of 16.6×, reflecting the improvement obtained by offloading the computational workload to the hardware accelerators while processing one kernel at a time. Conversely, the pipelined model reaches a 22.3× speedup, demonstrating the advantage of executing the five accelerators simultaneously. This result highlights how the pipelined approach maximizes resource utilization, further improving the overall system performance.

Finally, we report the resource utilization of the five accelerated kernels as percentage with respect to the available resources of the Kria KV260. The resource utilization takes into account i) the global memory and remote control architectural design; ii) the optimized version, named local memory and local control and iii) the whole

Table 1: Software execution time.

kernel	exec time [ms]
Gauss	**4.837**
cvt-color bgr2gray	**0.144**
Threshold	**0.027**
Filter2d	**0.294**
Mask	**0.141**
Hough	4.123
Lines separation	0.079
Regression	0.105
Predict turn	0.027
TOT	9.813

Table 2: Execution time and speedup achieved using Vitis Vision kernels.

	Global memory & remote control		**Local memory & remote control**		**Local memory & local control**	
kernel	**time**	**speedup**	**time**	**speedup**	**time**	**speedup**
Gauss	0.053	91.94×	0.053	91.94×	0.051	95.55×
cvt-color bgr2gray	0.177	0.814×	0.177	0.814×	0.175	0.823×
Threshold	0.027	1.000×	0.027	1.000×	0.025	1.080×
Filter2d	0.039	7.538×	0.038	7.737×	0.037	7.946×
Mask	0.054	2.611×	0.028	5.036×	0.025	5.640×
TOT (accelerated kernels)	0.350	15.65×	0.323	16.96×	0.313	17.50×
TOT (entire application)	4.684	2.087×	4.657	2.099×	4.647	2.104×

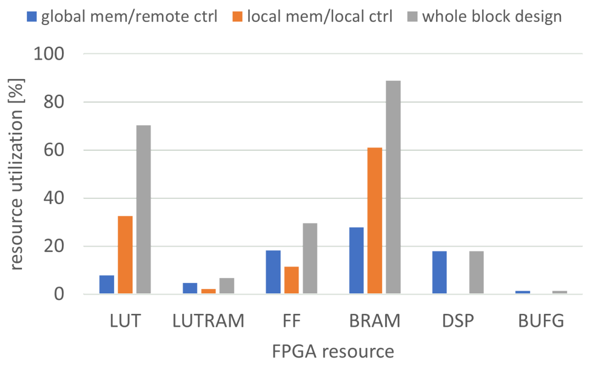

Figure 4: Resource utilization of *Global memory & remote control*, *Local memory & local control* versions.

block design implemented on top of the Kria SoC. As we can see from Figure 4, the final implementation uses ≈ 70% and ≈ 90% of LUTs and BRAMs respectively, while the utilization of the other resources is always less than 30%.

4 Conclusion

The results of this study highlight the effectiveness of FPGA-based hardware acceleration for lane detection tasks in autonomous vehicle applications. The proposed approach, implemented on the AMD Kria KV260, demonstrates that reconfigurable accelerators integrated within a Heterogeneous System on Chip (HeSoC) architecture can significantly improve performance. The experimental evaluation shows that the optimized integration architectures, memory organizations, and offloading strategies enable up to 22× reduction in latency compared to software-only implementations. These findings confirm the potential of FPGA technology to enhance computer vision pipelines, making it a compelling solution for high-performance and resource-efficient autonomous perception systems. As future work, we plan to extend the FPGA-based acceleration to the entire lane detection pipeline, including feature extraction and lane model fitting, to further improve performance and efficiency.

Acknowledgments

The project is supported by the Chips joint Undertaking and its members, including the top-up funding by the national Authorities of Germany, Belgium, Spain, Finland, Netherlands, Austria, Italy, Greece, Latvia, Lithuania and Turkey, under grant agreement number 101139996-2. Co-funded by the European Union. This work was supported by the European Union under the NextGenerationEU Programme within the Plan "PNRR - Missione 4 "Istruzione e Ricerca" - Componente C2 Investimento 1.1 "Fondo per il Programma Nazionale di Ricerca e Progetti di Rilevante Interesse Nazionale (PRIN)" by the Italian Ministry of University and Research (MUR)". Project Title: "Simplifying Predictable and energy-efficient Acceleration from Cloud to Edge (SPACE)", Project code: 202254AM7H, CUP: E53D23007810006, MUR D.D. financing decree n. 959, 30th June 2023.

References

[1] Mhd Rashed Al Koutayni, Gerd Reis, and Didier Stricker. 2023. DeepEdgeSoC: End-to-end deep learning framework for edge IoT devices. *Internet of Things* 21 (2023), 100665. doi:10.1016/j.iot.2022.100665
[2] AMD. 2025. AMD Kria KV260 Vision AI Starter Kit. https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html. Accessed: 2025-03-03.
[3] AMD. 2025. AMD Vitis Vision Library. https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vitis/vitis-libraries/vitis-vision.html. Accessed: 2025-03-03.
[4] AMD. 2025. Vitis AI. https://www.amd.com/en/products/software/vitis-ai.html. Accessed: 2025-03-03.
[5] Gianluca Bellocchi, Alessandro Capotondi, Francesco Conti, and Andrea Marongiu. 2021. A RISC-V-based FPGA Overlay to Simplify Embedded Accelerator Deployment. In *2021 24th Euromicro Conference on Digital System Design (DSD)*. IEEE, Vienna, Austria, 9–17. doi:10.1109/DSD53832.2021.00011
[6] Andrea Bernardi, Gianluca Brilli, Alessandro Capotondi, Andrea Marongiu, and Paolo Burgio. 2022. An FPGA Overlay for Efficient Real-Time Localization in 1/10th Scale Autonomous Vehicles. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Antwerp, Belgio, 915–920. doi:10.23919/DATE54114.2022.9774517
[7] Ziaul Choudhury, Anish Gulati, and Suresh Purini. 2023. FlowPix: Accelerating Image Processing Pipelines on an FPGA Overlay using a Domain Specific Compiler. *ACM Trans. Archit. Code Optim.* 20, 4, Article 60 (Dec. 2023), 25 pages. doi:10.1145/3629523
[8] Wencheng Han and Jianbing Shen. 2023. Decoupling the Curve Modeling and Pavement Regression for Lane Detection. arXiv:2309.10533 [cs.CV] https://arxiv.org/abs/2309.10533
[9] Yueru Luo, Chaoda Zheng, Xu Yan, Tang Kun, Chao Zheng, Shuguang Cui, and Zhen Li. 2023. LATR: 3D Lane Detection from Monocular Images with Transformer. arXiv:2308.04583 [cs.CV] https://arxiv.org/abs/2308.04583
[10] Ryo Nakagawa, Yoshiki Yamaguchi, Hajime Nobuhara, Iman Firmansyah, and Kai Sheng. 2024. Stereo Vision System for Crop Height Measurement Using a Monocular Camera-Equipped Drone. In *2024 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE, Las Vegas, Nevada, USA, 1–2. doi:10.1109/ICCE59016.2024.10444287
[11] Kamal Sehairi and Thierry Bouwmans. 2024. Accelerating Image Processing Functions on ZYNQ-FPGA Using the Vitis Vision Library. In *2024 3rd International Conference on Advanced Electrical Engineering (ICAEE)*. IEEE, Sidi-Bel-Abbès, Algeria, 1–6. doi:10.1109/ICAEE61760.2024.10783257
[12] Bobokhon Yusupbaev, Ke Yu, and Jun Rim Choi. 2024. Detection of Road Line Markings Based on Memory-Centric Computing. In *2024 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC)*. IEEE, Okinawa, Giappone, 1–4. doi:10.1109/ITC-CSCC62988.2024.10628362