Bringing Cloud-Connected Automotive Workloads to RISC-V: A CVA6-Based FPGA Case Study

Tianhai Liu[1]*, James J. Hunt[1], Holger Blasum[2], Darshak Sheladiya[2]

[1]aicas GmbH, Germany [2]SYSGO - Embedding Innovations, Germany

Abstract

An end-to-end case study evaluating cloud-connected workloads on CVA6 platforms is presented. System behaviour under increasing telemetry loads is analysed using CAN trace replay. The results provide empirical insights into the suitability of open RISC-V platforms for industrial deployment and highlight further optimisation.

Introduction

Automotive systems are increasingly cloud-connected, and RISC-V adoption in this domain continues to grow. Yet evidence that CVA6 platforms can support realistic end-to-end software stacks remains limited. Demonstrating this capability is key to evaluating their industrial readiness.

An end-to-end case study of the CVA6 core deployed on an FPGA platform is presented, complemented by a QEMU-based RISC-V environment and a native PC serving as correctness references. The study integrates an embedded Linux system, realtime Java middleware, and MQTT-based cloud connectivity.

The results provide initial empirical insights for the RISC-V and automotive communities regarding the integration of cloud-connected workloads on FPGA-based CVA6 platforms.

Automotive Use Case

Figure 1 describes a modern vehicle backend integration scenario in which a RISC-V-based edge device executes application logic locally and exchanges data with a cloud via an MQTT channel.

On the *edge side*, the system uses Java Runtime [1] which compiles the Java-based *Cloud Connector* [2] ahead of time into a standalone native executable for target platforms, in this case a RISC-V binary. This eliminates the need for a full JVM at runtime and provides predictable execution characteristics suitable for resource-constrained embedded environments. The Cloud Connector replays and aggregates vehicle signals from representative CAN traces, then transmits telemetry to the cloud via MQTT at configurable rates, packaging them into aggregated messages before transmission. On the *cloud side*, Dashboard [3] with AWS IoT Core is used to visualise the transmitted telemetry.

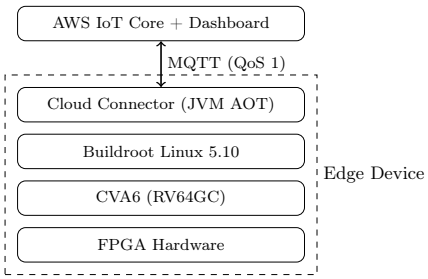


Figure 1: *Cloud–edge integration stack.*

Evaluation

The evaluation covers three execution platforms:

- Digilent Genesys 2 (Xilinx Kintex-7 XC7K325T) with a CVA6 (RV64GC) soft-core at 50–100 MHz, 1 GB DDR3, and Gigabit Ethernet, running Buildroot Linux (5.10). The 64-bit bitstream and images are built using the CVA6 project [4] and SDK [5].
- QEMU emulating an RV64GC platform, running Debian sid with Linux kernel 6.18, with 1 GB RAM.
- Ubuntu 24.04 running Linux kernel 6.17 on an Intel Core i7-class CPU at 3.3 GHz, with 32 GB RAM.

All platforms run identical AOT-compiled application logic using the Paho MQTT client on a single CPU core. Each configuration was executed 100 times, and the results are averaged to reduce variability.

Due to the inherent performance limitations of FPGA platforms, absolute benchmarking is not pursued. Instead, *functional feasibility* and *trend-based* behaviour under varying telemetry loads are analysed. Two workload dimensions are varied independently:

*Corresponding author: tianhai.liu@aicas.com

This work is partially funded by the CHIPS-JU project, TRISTAN, grant number 101095947.

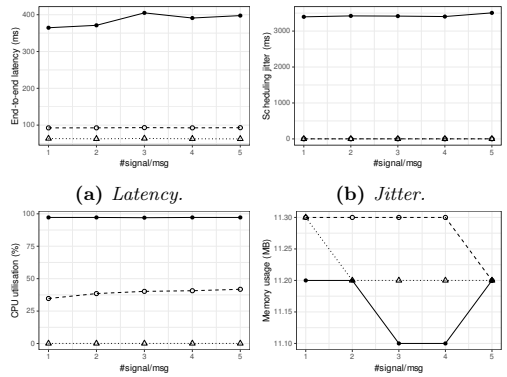
[1] https://www.aicas.com/products-services/jamaicavm

[2] https://github.com/tianhailiu/tristan-cloud-connector

[3] https://www.aicas.com/products-services/aicas-edge-data-gateway

[4] https://github.com/openhwgroup/cva6

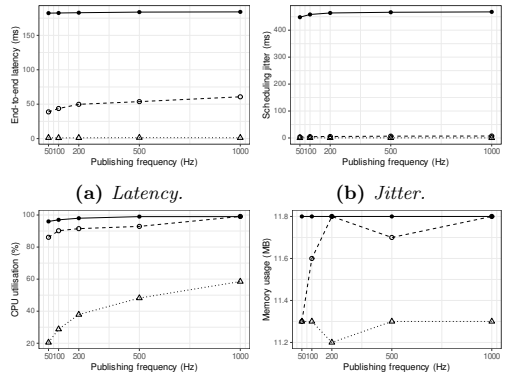
[5] https://github.com/openhwgroup/cva6-sdk



(a) *Latency.* **(b)** *Jitter.*

(c) *CPU.* **(d)** *Memory.*

Figure 2: *Trends with CAN signals per message. FPGA (solid), QEMU (dashed), PC (dotted).*



(a) *Latency.* **(b)** *Jitter.*

(c) *CPU.* **(d)** *Memory.*

Figure 3: *Trends with publishing frequency. FPGA (solid), QEMU (dashed), PC (dotted).*

(i) the number of aggregated CAN signals per MQTT message and (ii) the MQTT publishing frequency.

Figures 2 and 3 summarise the experimental results. Both figures report latency, jitter, CPU utilisation, and memory usage. *Latency* measures the time from message publishing to broker acknowledgement (`PUBACK`, QoS 1). *Jitter* is defined as the absolute deviation between the actual publishing time and the scheduled execution time of the publishing thread. *CPU* reflects process-level utilisation measured via the Linux `time` command, computed as (user + sys)/real time. *Memory* denotes peak Java heap size measured using RTSJ HeapMemory API.

Figure 2 reports the behaviour when publishing MQTT messages to Dashboard with 1–5 signals per message, at a fixed publishing frequency of 1 Hz. Each CAN signal contributes approximately 40 bytes to the message payload. During experimentation, the FPGA Ethernet subsystem showed recurring instability under sustained TCP traffic, affecting both MQTT and SSH/SCP transfers. When sending 5 CAN signals per message, 90% of runs failed due to connection loss after a single message (≈200 bytes), and higher numbers of signals were largely unsuccessful. This indicates a platform-level limitation in the current Ethernet IP and driver integration; therefore, higher numbers of signals per message are omitted from Figure 2.

Figure 3 instead evaluates the impact of publishing frequency while fixing each message to 40 signals. The publishing rate varies from 50 Hz to 1000 Hz. To avoid the Ethernet instability observed in the previous experiment, these extreme workload measurements were conducted using a locally hosted MQTT broker (Mosquitto), which isolates Ethernet and external network effects.

Both figures show that system behaviour remains stable with a small range and under extreme workloads, demonstrating feasibility. The FPGA platform exhibits relatively high jitter: nearly 3 times the scheduled time with internet access (Figure 2) and half without (Figure 3). This is likely due to the limited clock frequency of the soft-core CPU. Nevertheless, the jitter remains stable across workloads, indicating predictable, albeit coarse-grained, timing behaviour. Moreover, no data loss is observed under a stable Internet connection. Across all platforms, heap usage remains stable (≈11 MB), showing no workload-dependent growth as publishing frequency increases.

In particular, the reported end-to-end latency in Figure 2 intentionally includes public Internet effects, reflecting realistic industrial deployment conditions rather than laboratory isolation.

Overall, the results suggest the practical feasibility of integrating cloud-connected telemetry workloads on CVA6 FPGA-based platforms within the evaluated workload range. Scaling is primarily limited by processor capacity and scheduling, while memory usage and publishing frequency have negligible impact within the evaluated range. The observed Ethernet instability highlights the importance of network subsystem robustness for reliable high-load operation.

Conclusion

This work presents an empirical evaluation of the execution of cloud-connected automotive workloads on an FPGA-based CVA6 platform. The evaluation demonstrates practical integration and underscores the importance of strengthening Ethernet IP and driver support within the OpenHW ecosystem, which remains a key step toward production-grade deployment. Community contributions in this direction are welcome.

Future work will add TLS, integrate remote management and OTA updates [6] , and adopt a commercial Ethernet IP [7] to improve network stability.

[6] `https://www.aicas.com/products-services/aicas-edge-device-portal`

[7] `https://www.xilinx.com/products/intellectual-property/axi_ethernet.html`