(GDPR) must be detailed because they are essential for organizations to identify and secure the data effectively and efficiently.[3]

Moreover, data protection can be strengthened by visualizing and mapping network connections through a conventional architecture diagram, which illustrates the paths of data flow. The use of a standard network diagram to map connections and trace data flow is a foundational step in ensuring data protection.

In conclusion, Zero Trust Architecture represents a paradigm shift in cloud security, replacing implicit trust with continuous, context-aware verification. Its principles—centered on identity, least privilege, and adaptive policy enforcement—align well with the challenges of cloud environments. Although implementing ZTA presents technical and organizational challenges, the benefits in terms of security posture, attack containment, and compliance make it a compelling framework for modern cloud-centric infrastructures. Continued research and technological evolution will further strengthen its adoption and effectiveness.

Resources

1. Scott Rose, Oliver Borchert, Stu Mitchell, Sean Connelly. 2020. Zero Trust Architecture. NIST Special Publication 800-207. DOI: https://doi.org/10.6028/NIST.SP.800-207
2. Akharchaf, Youssef. (2025). Zero Trust Architecture in Cloud Security: Challenges and Implementation. 10.13140/RG.2.2.26671.04000.
3. Ahmadi, Sina. (2024). Zero Trust Architecture in Cloud Networks: Application, Challenges and Future Opportunities. Journal of Engineering Research and Reports. 26. 215-228. 10.9734/JERR/2024/v26i21083.

DOI 10.70286/ISU-04.02.2026.010

EVALUATING THE IMPACT OF SOFTWARE-BASED ECC MEMORY PROTECTION ON THE PERFORMANCE OF UAV ONBOARD SOFTWARE

Nyzhnyk Andrii
PhD student
Department of Information Security
Lviv Polytechnic National University, Lviv, Ukraine

Abstract. This paper evaluates the impact of software-based ECC protection of volatile memory on the performance of onboard software for unmanned aerial vehicles (UAVs). The proposed approach computes and verifies check bits at the level of software containers for critical data structures using a SECDED-class code (single-bit error correction and double-bit error detection).

The experimental evaluation is carried out using microbenchmarks that reproduce memory-access patterns typical for onboard systems: periodic updates of a small state (read–modify–write), streaming operations over telemetry/sensor buffers, random reads from tables, and a mixed workload. Measurements are performed in a release build with warm-up for representative working-set sizes (state: 2 KiB, buffer: 8 MiB, table: 8 MiB).

The results show that the memory overhead of software ECC is stable and predictable at approximately 12.5% (1 byte of check data per 8 bytes of payload). At the same time, the time overhead depends on the access pattern and is substantial in the current implementation. Specifically, the measured slowdown factors are approximately 180 times for small-state updates, 654 times for the streaming mode, 26 times for random reads, and 73 times for the mixed workload.

These findings indicate that applying software ECC to large streaming buffers without additional optimization may be impractical for real-time systems, whereas selective protection of the most critical structures is more feasible in practice.

Keywords: UAV; onboard software; reliability; random-access memory; ECC; SECDED; software data protection; Rust; microbenchmarks; performance.

Introduction. Modern UAV onboard computing systems perform navigation, stabilization, trajectory planning, and sensor data processing in real time. Memory errors that occur without physical damage to components (soft errors) [1] can distort the states of estimation filters, control parameters, or intermediate computation results and, as a consequence, lead to degraded accuracy or emergency operating modes [2].

One typical source of such transient faults is single-event upsets (SEUs), caused, for example, by cosmic radiation, electromagnetic interference, or operating conditions [3]. Hardware ECC (Error-Correcting Code) is widely used in systems where silent data corruption is unacceptable: it can automatically correct single-bit errors and detect double-bit errors (the SECDED approach - Single-Error Correction, Double-Error Detection) [4-5]. However, a significant portion of embedded platforms used in UAVs (especially low-cost or energy-constrained ones) do not have full-featured ECC memory or do not provide transparent ECC for all critical data [6].

In this context, a software ECC approach is of practical interest: adding check bits and verification/correction procedures at the level of software data structures and critical buffers [7]. Such protection does not fully replace hardware ECC, but it can significantly increase the robustness of specific components of UAV onboard software against transient memory faults by enabling a controlled response (correction, detection, and signaling of uncorrectable errors) [8].

The aim of this work is to quantitatively evaluate how software ECC protection of critical data structures affects performance (latency/throughput) and memory consumption in workloads typical of UAV onboard software. Prototypes and microbenchmarks are implemented in Rust, which provides strict memory-safety guarantees and convenient mechanisms for modeling low-level data structures important for real-time systems.

The tasks addressed in this work are:

1. Identify the class of critical onboard-software data for which RAM errors pose the highest risk (navigation/filter state, control parameters, sensor data buffers, configurations).

2. Implement two variants of data storage/access:

a. Baseline: standard data structures without protection;

b. Protected: data structures with ECC redundancy (SECDED-class: single-bit correction and double-bit detection) implemented at the level of software containers in Rust.

3. Measure the slowdown (latency/throughput) for representative data-access patterns.

4. Estimate memory overhead:

a. structural (type sizes);

b. allocation/actual (for buffers/vectors).

Evaluation Methodology. To assess the impact of software ECC protection, a comparative experiment was conducted using two data-access implementations: (i) baseline data structures without protection and (ii) data structures with software SECDED-type ECC (single-bit error correction and double-bit error detection). Both variants execute identical sequences of read/write operations, ensuring a fair comparison.

Performance is evaluated with microbenchmarks that reproduce memory-access patterns typical for UAV onboard software:

- periodic updates of a small state (read–modify–write);
- sequential processing of a telemetry/sensor-data buffer (stream access);
- pseudo-random reads of small blocks (lookup access).

For each pattern, execution-time metrics (ns/op or ops/s) are recorded in a release build after warm-up and with repeated runs. Memory overhead is defined as the ratio of protected-structure sizes to baseline sizes (*size_of*), and for buffers as the additional volume required to store ECC check bits per data block.

Experimental Results. The aggregated performance results are presented in Table 1. As shown in Table 1, the software ECC in the current implementation introduces a substantial time overhead across all access patterns. The largest slowdown is observed in the streaming mode (P2), where baseline access is *cheap* and well optimized by the cache subsystem, while ECC encoding/verification costs dominate the overall execution time. The smallest slowdown is obtained for random reads (P3), where a significant portion of time is determined by memory-access latency.

Fault injection was not enabled in this run (corrected=0, detected_double=0); therefore, the results in Table 1 correspond to the *normal* ECC operating mode without actual corrections.

Table 1 - Performance metrics (ns/op) for baseline and protected (SECDED)

Pattern	Description	Baseline, ns/op	Protected, ns/op	Slowdown factor (protected/baseline)
P1	Small-state update (read–modify–write)	14.591	2626.503	180.01
P2	Streaming access (ring buffer)	1.594	1042.816	654.21
P3	Random reads (table lookup)	45.875	1191.022	25.96
P4	Mixed workload (state + ring + table)	19.674	1445.384	73.47

The memory-overhead assessment is given in Table 2. According to Table 2, the memory overhead is stable and equals +12.5% in all scenarios, which matches the SECDED approach for 64-bit blocks (adding 1 byte of check data per 8 bytes of payload). The VmRSS value within a single run was not used as the primary metric, because it is affected by the allocation life cycle during the sequential execution of baseline and protected variants; therefore, buf_bytes was used for quantitative comparison, as it directly reflects the allocated buffer size.

Table 2 - Memory overhead (buf_bytes) for baseline and protected (SECDED)

Pattern	Baseline, buf_bytes	Protected, buf_bytes	Δ, bytes	Δ, %
P1	2048	2304	256	12.5
P2	8 388 608	9 437 184	1 048 576	12.5
P3	8 388 608	9 437 184	1 048 576	12.5
P4	4 196 352	4 720 896	524 544	12.5

Limitations of the Experiment and Directions for Optimisation. The presented results should be interpreted with several limitations in mind. First, the implemented software ECC operates as a *wrapper* over the data and performs verification/encoding on every access to a protected element; this model is useful for estimating an upper bound on overhead, but it does not replicate hardware ECC, where check-bit computation occurs in parallel with memory access and has a much smaller impact on latency.

Second, the study relies on microbenchmarks that reflect typical access patterns (state updates, streaming, and random access), but they do not cover the full spectrum of UAV onboard workloads (e.g., sensor processing, trajectory planning, vector-algebra computations), where the share of computation may be higher and the relative ECC impact lower.

Third, fault injection was not enabled in the reported measurement series; therefore, the results characterize the “normal” operating mode (verification without actual corrections) and do not capture the effect of corrections on the tail of the latency distribution.

Conclusions. This work considered a software approach to ECC protection of critical data in the memory of UAV onboard software and experimentally evaluated its impact on performance and memory costs. The conducted microbenchmarks for typical access patterns (small-state updates, streaming buffer processing, random reads, and a mixed workload) show that a SECDED-class ECC implementation at the level of software containers provides a predictable and stable memory overhead but may introduce substantial time overhead.

The experiment confirmed that the memory overhead matches the expected SECDED estimate for 64-bit blocks and equals approximately 12.5% in all scenarios. At the same time, the time overhead strongly depends on the memory-access pattern: the execution time increased by a factor of approximately 25.96 for random reads from a large table, 180.01 for small-state updates, 73.47 for the mixed workload, and 654.21 for streaming access to a buffer. Thus, in the current implementation, ECC overhead becomes the dominant factor in scenarios where baseline memory-access operations are cheap and cache-friendly.

Based on these results, software ECC is most appropriate for UAVs as a selective mechanism applied to a limited set of highly critical data (navigation/control state, safety parameters, configurations), whereas applying ECC to large streaming telemetry and sensor buffers without additional optimization may be impractical for real-time systems.

Promising directions for future work include optimizing encoding/verification algorithms, reducing the frequency of ECC operations (periodic scrubbing and checks at control-loop boundaries), and combining ECC with lighter integrity mechanisms for non-critical streaming data.

References

1. Wang, W., Li, X., Chen, L., Sun, H., & Zhang, F. (2024). A review on soft error correcting techniques of aerospace-grade static ram-based field-programmable gate arrays. Sensors, 24(16), 5356.
2. Ahmed, F., & Jenihhin, M. (2022). A survey on UAV computing platforms: A hardware reliability perspective. Sensors, 22(16), 6286.
3. Abich, G., Garibotti, R., Reis, R., & Ost, L. (2022). The impact of soft errors in memory units of edge devices executing convolutional neural networks. IEEE Transactions on Circuits and Systems II: Express Briefs, 69(3), 679-683.
4. Sridevi, N., Jamal, K., & Mannem, K. (2021, April). Implementation of error correction techniques in memory applications. In 2021 5th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 586-595). IEEE.
5. Nezzari, Y. (2020). Improving processor reliability using software protection techniques (Doctoral dissertation, University of Surrey).

6. Wilson, A. N., Kumar, A., Jha, A., & Cenkeramaddi, L. R. (2021). Embedded sensors, communication technologies, computing platforms and machine learning for UAVs: A review. IEEE Sensors Journal, 22(3), 1807-1826.
7. Buch, S. (2023, October). Error detecting and correcting codes for dram functional safety. In Proceedings of the International Symposium on Memory Systems (pp. 1-5).
8. Alam, I., Dolecek, L., & Gupta, P. (2021). Lightweight software-defined error correction for memories. Dependable Embedded Systems, 207.