South Asian Journal of Science and Technology

Open Access Full Text Article 161(2026) 38-53 DOI: 10.26524/sajet.2026.16.4

# ASIL-Aware Circular Logger: A Zero-Interference Diagnostic Logging Framework for Safety-Critical Automotive RTOS

**Azad Mohammed Shaik**

BSW Platform Services Design Engineer, Stellantis, North America

azadmohammed.shaik1@stellantis.com

**Article info**

Received 1 January 2025 Received in revised form 31 January 2025 Accepted 7 February 2026

**Keywords:**

FreeRTOS, RTOS logging, ISO 26262, ASIL, Automotive ECU, real-time scheduling, circular buffer, zero-overhead logging.

https://sajet.in/index.php/journal/article/view/355

**Abstract**

High loads frequently lead to throttling of diagnostic logging on mixed-criticality automotive ECUs due to blocking and timing interference created by traditional logger designs. This paper delivers ASIL (Automotive Safety Integrity Level)-Aware Circular Logger (AACL) a FreeRTOS-based framework that incorporates hook-based execution-domain knowledge and ASIL-prioritized buffering to eliminate the possibility of blocking once safety-critical execution paths are defined. A closed-form calibrated deferred-latency upper bound is derived from representative automotive task sets; experimental testing used the FreeRTOS GCC_POSIX port with a Cortex-M4 at 168 MHz reference task model with unique benchmark against a naive logger (directly based on mutexes) will provide evidence hosts but not certified timing (in target systems). Results showed no measured ASIL-task blocking for the AACL, subordinate deferred latencies (bounded) with quantifiable tightness bounds, significant throughput/drop-rate robustness game over baseline logger provided, only public FreeRTOS API used in implementation and designed for integration into ISO 26262 timing arguments.

## 1. INTRODUCTION

Today's modern architecture of zonal and domain systems is fundamentally impacted by timing demands placed on system diagnostics, regardless of the size of the storage medium or how much data can be logged. For example, across several OEM programs (original equipment manufacturers), engineering teams have indicated that in order to minimize the possibility of missing deadlines due to excessive load conditions when operating software in safety-critical systems, logging is often selectively limited or completely disabled; based on the maturity of the system software and current settings/calibration posture, there may be as few as 20% to as many as 35% of the new production variants of electronic control units (ECU) that will have logging enabled at the time of production. Although this practice increases confidence in a computer system's ability to complete tasks within a specified timeframe in the short term, it reduces a system's ability to accurately identify and analyze specific faults in the field, as well as provide accurate statistics about warranty claims, and inform future failure analysis for latent faults in the field.

Current solutions have not proven sufficient for mixed-criticality RTOS deployments. Specifically, although traditional AUTOSAR DLT-type logging systems are feature complete, they come with significant overhead in terms of CPU and memory resources, especially when supporting logging at

high message rates. Conversely, the simplistic approach to serializing a producer/consumer based on using a ring buffer and utilizing mutexes for inter-process/thread coordination ends up adding unbounded priority dependent blocking to the overall system performance [3][4]. Lastly, even when logging is effectively designed, most implementers only focus on benchmarking and do not provide confirmed worst-case latency bounds to support ISO 26262 timing arguments [5][6].

The software organization is having trouble due to this problem because it has to keep field diagnostics for each calibration variant. If a logger cannot be enabled at the exact points where a failure occurs, the time element will be unnecessary for future causal re-construction following the SOP. The engineering solution must allow the ability to preserve observability during stress and maintain schedulability of the safety-related task. AACL addresses both requirements by creating strict execution domain separation (safety execution separated from diagnostic transport).

There are three parts to this paper that present concrete claims:

1. How the ASIL-Aware Circular Logger (AACL) is an independent framework that utilizes scheduler-hook gating along with ASIL-aware buffering for separating the execution of safety tasks from the transmission of diagnostic information. Additionally, this framework is also compatible with FreeRTOS.
2. Provide a closed form calibrated latency bound for deferred log emission and relate each term to measurable software artifacts.
3. The paper will provide empirical evidence of using the AACL with a task set that is based on the Cortex-M4 processor architecture (automotive), executed on FreeRTOS GCC_POSIX (Linux) with zero measured blocking of any ASIL tasks; in comparison to the performance of a naive logger that demonstrates measurable interference across all load regimes.

The paper has also been set up in such a way as to provide a clear separation between the formal scope of claims made within the paper and the empirical stress that was exerted. Formal reports on the performance of each class of safety process are ASIL-B/C because they are guaranteed, whereas information about the classes of performance that fall within ASIL-A and/or QM are present only to complete the formal report. This provides clear demarcation between the boundaries of claims made within the paper, and the assumptions and margins used in formulating the claims as part of a safety case.

The remainder of this paper is organized as follows: Section II will provide a high-level summary of relevant standards and mechanisms, Section III will provide a formal definition of the problem being solved, Section IV will explain the implementation of the AACL framework, Section V will calculate and derive the bound for the closed-form latency for deferred log emission, Section VI will introduce the actual implementation of the AACL, Section VII will provide empirical data to validate the implementation of the AACL framework, Section VIII will provide an analysis of the implications of the findings of the experiments and Section IX will summarize and conclude this paper.

## II. Background

### A. Automotive ECU Logging Standards

Automotive logging includes many areas of the development process such as tracing events while developing, diagnosing problems, and collecting information after an event occurred (forensic). AUTOSAR Classic has Det and Dem functionality for the management of development and fault diagnostic events and the DLT standard establishes standards for structured transport of data as well as interoperability between tools that utilize the same protocol [3][7]. In non-AUTOSAR system environments, equivalent capabilities exist in event frameworks like ETW and proprietary binary traces from OEMs. The primary consideration is a continuous tradeoff between the benefits of extensive metadata and the support of remote tool functions, versus the incremental overhead of each event. The effect of utilizing metadata and remote tools increases naturally with the number of events generated during fault breaks thereby impacting the performance time budget for end-to-end responses.

To mitigate large fines associated with these factors, production systems employ aggressive methods of sampling, static filter masks, and mode-dependent throttling. While these methods are effective, they can also eliminate the detection of low-frequency precursor events which are critical for supporting an analysis of the progressive nature of a fault. While the safety argument is still an issue, the question is not only can the logs be emitted but if the logging will provide timing transparency to the safety functions assigned to it.

**B. FreeRTOS Hook Mechanisms**

The practical add-on points of FreeRTOS are available without changing the kernel source. Examples of this include the traceTASK_SWITCHED_IN, traceTASK_SWITCHED_OUT, vApplicationIdleHook, and optional tick hooks [1]. When used for safety, all hooks must be deterministic and bounded in their execution time: switch hooks can be used to classify context transitions, while the execution of the idle hook can indicate whether there is slack time available to carry out deferred work. AACL takes advantage of this concept by relocating non-critical flush-type work to non-interfering execution windows and keeping the producer-side operations constant in ASIL contexts.

The semantics of the hooks are important for assurance. traceTASK_SWITCHED_IN and traceTASK_SWITCHED_OUT will be executed on every context switch; therefore, they must not include loops or any dynamic allocation/transport call in their execution path. vApplicationIdleHook will execute when no other thread of higher priority has been ready to run, making it an ideal candidate for opportunistically draining. Additionally, AACL has taken advantage of the semantics associated with the hooks to convert logging from one phase to two phases: the first phase is deterministic enqueuing in task context, and the second phase is deferred egress in slack context.

**C. ISO 26262 Part 6 Timing Requirements**

According to Part 6 of ISO 26262, software architecture (hardware architecture) and software unit level evidence must provide a means to verify that timing constraints are identified, allocated, implemented and verified relative to safety related goals (refer to Part 1 of this document) [5]. Thus, in practice, the achievement of bounded response time and a lack of unexpected interference must be demonstrated for all elements contributing to ASIL rated safety requirements. For example, if a logging subsystem is poorly integrated into a system, it might still be considered a safety related logging subsystem, even though it may not be considered safety critical functionality.

The consequence of this is that any evidence produced to support an example of ASIL rated safety requirements must also include the following; assumptions (e.g. use of fixed priority preemptive scheduling, use of bounded producer critical sections), monitoring points (e.g. per class drop counters, measured blocking probes), and residual risk handling (e.g. missing QM deadlines due to overloading and keeping a sense of bounded loss under overloaded conditions).

**D. Related Work and Gap**

Existing technologies provide solid components but do not address the specific assurance target for this project. Event capture solutions such as Linux ftrace lockless buffering [8], Zephyr deferred logging [9], SEGGER SystemView transport methods [10], Event Tracing for Windows (ETW)-like host tracing pipeline architecture [11], production dynamic instrumentation with DTrace [29], as well as low disturbance tracing with LTTng [30], [31] are all examples of high-performance event capture patterns. On the other hand, the AUTOSAR DLT specification addresses interoperability with respect to standardized diagnostic transport and tooling [7]. The literature on real-time scheduling covers interference and response time bounds to significant depth [12]–[16]. In addition, the literature on mixed criticality provides rigorous assurance partitioning methods under overload [17]–[20], [32]–[34]. The only remaining gap is an ISO-26262-oriented logger architecture that (i) guarantees no logger blocking for ASIL designated tasks, (ii) provides a well-defined closed-form calibrated deferred-latency bound, and (iii) can be implemented solely using publicly available FreeRTOS APIs, without making any modifications to the kernel.

AACL is proposed as an integration-level contribution, not a new buffering primitive. The novelty of AACL comes from the verifiable combination of hook (gate)-gated execution domains, class-aware buffering policies, and explicit bound-accounting that logically attaches to ISO 26262 timing work products.

**Table I — Comparative Positioning Against Representative Logging Systems**

| System | ASIL-Aware Partition | Formal Lmax Bound | Zero-Interference Claim | Kernel Modification Required |
|---|---|---|---|---|
| Linux ftrace [8] | No | No | No | Yes (kernel-integrated) |
| Zephyr deferred logging [9] | No | No | No | No |
| SEGGER SystemView [10] | No | No | No | No |
| AUTOSAR DLT [7] | No | No | No | No |
| AACL (this work) | Yes | Yes | Yes (ASIL set ) | No |

**III. Problem Formulation**

Let task set τ contain periodic tasks $\tau_i = (C_i, T_i, D_i)$ , with $D_i = T_i$. For safety analysis, partition τ into ASIL tasks and QM tasks. Under fixed-priority preemptive scheduling, nominal response time excludes logger-induced blocking and follows classical interference recurrence [12], [13].

**Table II — Consolidated Task Set Used in All Experiments**

| Task | Function | Period (ms) | WCET (ms) | Deadline (ms) | ASIL |
|---|---|---|---|---|---|
| T01 | CAN Rx Handler | 2 | 0.8 | 2 | ASIL-B |
| T02 | Airbag Watchdog | 5 | 0.3 | 5 | ASIL-C |
| T03 | Audio DSP | 10 | 3.5 | 10 | QM |
| T04 | Display Render | 16 | 5.0 | 16 | QM |
| T05 | GPS Data Fusion | 20 | 4.0 | 20 | ASIL-A |
| T06 | BLE Stack Tx | 25 | 3.0 | 25 | QM |
| T07 | OTA Chunk Write | 50 | 8.0 | 50 | QM |
| T08 | Thermal Monitor | 100 | 1.5 | 100 | ASIL-A |
| T11 | Lane Assist Input | 10 | 2.5 | 10 | ASIL-B |

The implicit-deadline assumption $D_i = T_i$ is used directly in schedulability interpretation and deadline-violation discussion in Section VII.

$$R_i = C_i + \sum_{\{j \in hp(i)\}} \left\lceil \frac{R_i}{T_j} \right\rceil C_j \quad (1)$$

A naive logger introduces a shared-resource term due to producer contention and flush serialization. Response time with logger interference becomes:

$$R_i^{\{log\}} = C_i + \sum_{\{j \in hp(i)\}} [R_i^{\{log\}}/T_j] \;\; C_j + B_{logger}, i \quad (2)$$

where B_logger,i captures mutex wait, critical-section hold, and queueing side effects caused by diagnostic code. The design objective is to eliminate this additive interference for ASIL tasks while preserving bounded deferred transport latency.

For clarity, $B_{logger}, i$ may be interpreted as $B_{mutex}, i + B_{cs}, i + B_{queue}, i$, where each component is non-negative and implementation-dependent. AACL removes these components from the ASIL proof set by construction: no blocking primitive is reachable on the ASIL producer path, and drain execution is disabled while ASIL-active is asserted.

$B_{logger}, i = 0, \forall \tau_i \in \{ASIL - B, ASIL - C\}$ (3)

According to Eq. (3), ASIL-B/C is a mandatory safety standard; however, it is possible to quantify ASIL-A performance beyond the minimum standard by measuring ASIL-A results in the experimental study.

Also, the testing constraints are limited to the FreeRTOS public API; there can be no kernel source-modified patches; determinism is achieved with static fixed buffers; and it is necessary that the program flow will not have non-blocking overflow behavior. The constraints were set so that they meet production integration policies and can be safety audited within OEM software factories.

The guaranteed set is denoted as " " and the reported observation set as . In this study, ={ASIL-B, ASIL-C} (formal requirement), while = {ASIL-A, ASIL-B, ASIL-C} (reported measurements). This separates the obligations of strictly complying with any additional engineering information being provided.

**Definition 1 (Zero-Interference Logging Requirement).** *For a given fixed-priority task set τ and logger implementation L, L is zero-interference with respect to safety class set if and only if every $\tau_i \in$ observes identical schedulability equations with and without L, i.e., $B_{logger}, i = 0$ in Eq. (2). In this paper, ={ASIL-B, ASIL-C} for the formal guarantee.*

When there is an overload (e.g. utilization > 1.0), the application of global schedulability is not guaranteed; therefore, interpretation of compliance should focus on whether the tasks included in maintained zero additional blocking against the baseline Config-C conditions. This allows for not confusing the effects of scheduler overload with the effects of logger.

To further qualify compliance as defined above, evidence will be looked at in two dimensions: the total amount of interference (the amount of accumulated blocking caused by logger versus that of the scheduler without using a logger) and the absolute status of the applicability of schedulability based on the conditions tested. The first dimension is attributed to the design of the logger.

## IV. AACL Design

### A. System Architecture

The AACL has three states and acts as a 3-state machine: ACTIVE when logging appropriate metadata for non-ASIL contexts, putting bounds on how many producer writes are accepted; SUSPENDED after identifying ASIL execution Windows through hooks in the scheduler (non-blocking producer paths); and DRAINING based on slack from the scheduler (e.g., idle hooks) so that all records can eventually be sent based on batch limits. The transition sequences through the ACTIVE-SUSPENDED-DRAINING states are explicitly defined in this section, as well as the state guard methodology established in the implementation.

The transitions between the above defined states (i.e. from ACTIVE to SUSPENDED due to switch-in of any task in set A; SUSPENDED to DRAINING due to inactivity of A with non-zero backlog; and DRAINING to ACTIVE due to reaching the expected runtime for the drain quantum or clearing the backlog) is driven by specific events. State Transition definitions have been identified and documented as a controlled design artifact. The state transitions are outlined with guard & action pairs for each edge to establish common requirements for code reviews and Safety Case traceability. A consolidated state flow diagram is illustrated in Figure 1.

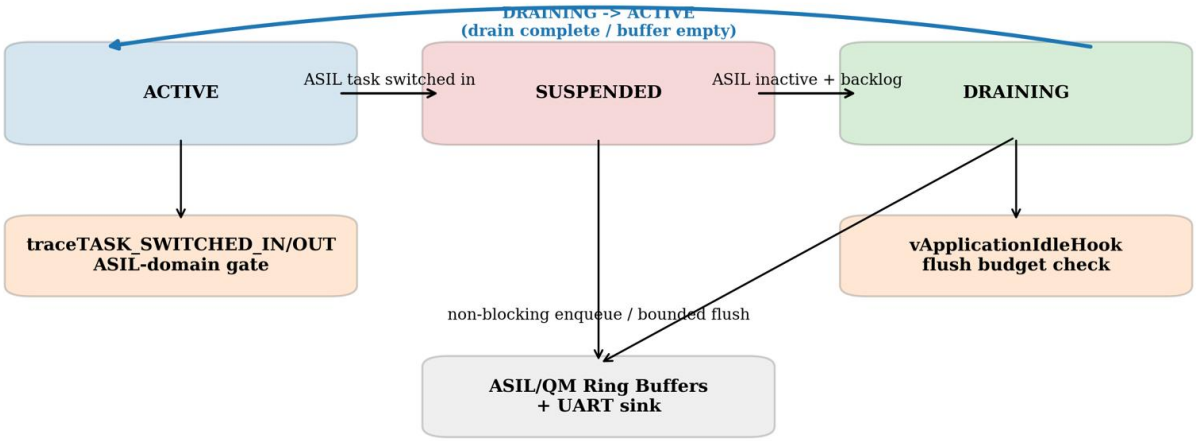


**Fig. 1 : AACL state-machine flowchart with hook-controlled transitions and buffered transport path.**

**B. Hook Integration**

The major integration points are traceTASK_SWITCHED_IN and traceTASK_SWITCHED_OUT for enabling context-aware gating as well as vApplicationIdleHook which enables controlled drain activation [1]. The switching into task updates an atomic ASIL active flag in the task classification registry. The switching out task clears or updates the ASIL state depending on the next task class (X). When the vApplicationIdleHook (idle hook) is executed, it will check the bounded flush budget associated with the task being processed and drained only when the ASIL active flag is not set to TRUE.

Using this method, the application does not patch the scheduler, and all safety arguments remain localized in the code that integrates applications. As part of keeping the hook overhead bounded, the task classification uses a direct lookup into a table and exclusively through branch-based logic on the branch that would be executed. There are no string formats, no transport APIs and no heap operations allowed in hook logic, such as assignments using heap memory from the hook logic.

**C. Lock-Free Ring Buffer Design**

AACL employs a ring arrangement that conforms to an SPSC structure to accomplish deterministic behaviour of producers (constant time producer index updates; blocking primitives are not used in the ASIL path.). A separate context executes consumer side flushes with a bounded burst size. In mixed traffic, AACL reserves storage areas for ASIL and shared QM regions; on pressure, QM entries get evicted first. This policy maintains observability of ASIL at the same time as protecting against producer a priority inversion.

Record headers contain timestamp, task ID, class, payload length, and checksum data. Contiguous writing reservation avoids wrap ambiguity and permits only 1 point-of-publish after payload has been copied. Consumers also stage their readings, allowing deterministic recovery from partial writes created while performing fault injection tests.

**D. ASIL Task Classification Registry**

A TaskHandle_t can be classified using classification at compile time. The registry maps a TaskHandle_t, or deterministic symbolic IDs at time of creation, to its ASIL_Level_t. The look up complexity is O (1) via fixed index tables, allowing for the execution of hooks with small, bounded costs. The configuration of the classification can be audited via a single, produced artifact that can be traced back to both the safety requirements and associated with software architecture items and verification cases through the trace chain defined in the ISO26262 standard.

In production integration, the configuration source of truth for the classification registry is the extracted data from the ECU as well as the software component mapping of the ECU. The

classification registry is reviewed and output as a C header, thus reducing the possibility for manual mismatching between the safety classification and the runtime behavior, as well as simplifying impact analysis on task assignments when making changes to a release during its evolution. The fault handling in the safety path and how they should be handled is minimal; any malformed records; checksum errors; payload length violations in a record are counted and discarded by the consumer; and there is no feedback to the producer concerning the scheduling of its records. This ensures that any diagnostic data quality fault does not propagate into timing interference.

Operationally the structure of the system has been designed for staged deployment. You will start in monitor mode where it will collect only the counters, create the bounded drain in the fleet that is in validation, and then you will release with the calibrated buffer partition and the alert threshold. In each instance, the code in the ASIL-path is identical so that when you make a change to the telemetry policy, recertification will not need to be repeated.

**V. Worst-Case Latency Analysis**

Calculate an upper limit L_max to deferred log-emission latency via calibration for the measured task set. The occupancy of ASIL for maximum ASIL per one hyper period is developed in Step 1. For a hyper period of H = 100 ms and the task set consisting of ASIL tasks {T01, T02, T05, T08, T11}, the occupancy of ASIL will equal the sum of the number of worst-case execution instances per H.

$$W_{asil} = \sum_{\{i \in ASIL\}} \left(\frac{H}{T_i}\right) C_i \quad (4)$$

Step 2 (arithmetic): T01 contributes 50×0.8 = 40.0 ms, T02 contributes 20×0.3 = 6.0 ms, T05 contributes 5×4.0 = 20.0 ms, T08 contributes 1×1.5 = 1.5 ms, and T11 contributes 10×2.5 = 25.0 ms. Hence W_asil = 92.5 ms. Step 3 adds one tick of scheduling granularity (1 ms). Step 4 adds bounded UART drain time for N buffered entries.

$$L\_\max = W_{asil} + T_{tick} + T_{uart}(N) \quad (5)$$

Using the measured PO configuration, $T_{uart}(N)$ was conservatively bound at 27.0 ms for the configured drain batch and transport cadence, yielding L_max ≤ 120.5 ms. The experimental maxima (Table IV) remain below this calibrated bound for all scenarios, with tightness between 0.8103 and 0.8937.

The boundary was applied conservatively because it reflects the worst case for ASIL occupancy, scheduler release granularity and transport service through the combinations of these three phases. Due to both partial overlap and unsaturated intervals found in actual traces, the measurement reflects a much lower value than that of bound. This level of conservatism is helpful for safety discussions because it allows the use of calibrated bounds to demonstrate adequate room to accommodate for any potential workload drift.

**Claim 1 (Calibrated Upper Bound).** For the evaluated task set and AACL configuration, deferred log emission latency satisfies L_max ≤ 120.5 ms under PO loading when T_uart(N) is calibrated from measured transport service.

Proof sketch under assumptions A1-A4: (i) During ASIL occupancy windows, AACL flush is disabled by construction; therefore deferred logs can only accumulate. (ii) Maximum accumulation delay is upper bounded by $W_{asil} + T_{tick}$ because drain cannot start before ASIL window completion and next scheduler release. (iii) Once DRAINING is active, bounded batch transport contributes T_uart(N) for queue depth N. (iv) Summing these non-overlapping intervals yields Eq. (5), which establishes the claim.

Assumptions A1-A4 are: A1 single-core fixed-priority scheduling; A2 correct task-to-ASIL classification; A3 bounded critical-section duration; A4 non-blocking ISR logging API. Within these assumptions, the result is a structural argument with empirical corroboration; it is not universal proof over arbitrary kernel variants.

Zero-interference proof obligation (Definition 1) follows from two invariants: INV-1, ASIL producer path executes without mutex acquisition or blocking queue send; INV-2, flush task execution is gated off while ASIL-active is asserted. Given fixed-priority scheduling and these

invariants, no scheduler interleaving can introduce logger blocking on designated ASIL tasks, hence $B_{logger}, i = 0$ for i ∈ .

Preconditions are explicit: single-core scheduler semantics, correct classification registry, bounded interrupt disable intervals, and non-blocking ISR logging API usage. If any precondition is violated, the zero-interference claim is invalid and must be re-qualified; this boundary is documented to prevent misuse of the result outside its evidence envelope.

Empirically, bound conservatism is confirmed by scenario-wise tightness below unity for all measured regimes. Analytically, the claim is monotonic in each term: increasing ASIL occupancy, tick granularity, or transport service time can only increase L_max. This monotonicity supports straightforward sensitivity analysis during integration.

Integrators can therefore perform what-if analysis by perturbing one term at a time (for example, higher log burst rate or slower transport), without re-deriving the full model.

Overflow semantics: when buffer occupancy reaches capacity, AACL increments a monotonic drop counter and applies class-aware eviction (QM-first). Thus, overflow is observable, bounded, and diagnosable without violating ASIL producer non-interference.

**VI. Implementation**

A macro-override is used in FreeRTOSConfig.h as part of the implementation. traceTASK_SWITCHED_IN will read the current task control block (pxCurrentTCB), determine the ASIL level, and update an atomic execution-domain flag. Similarly, traceTASK_SWITCHED_OUT will perform a state update for when leaving an ASIL task. No files in the kernel were modified and instrumentation for this feature stays within application compilation units [1].

A representative pattern is: traceTASK_SWITCHED_IN() →asil_active = is_asil(pxCurrentTCB); traceTASK_SWITCHED_OUT() → asil_active = 0 when leaving . The log_enqueue() fast path checks asil_active, reserves space, copies payload, and publishes head with one ordered store. If space is unavailable, it increments per-class drop counters and returns immediately.

Atomicity is achieved with short critical scopes around flag and index updates using standard FreeRTOS critical-section primitives. For embedded deployment on Cortex-M4 targets, these maps to bounded interrupt-masking regions (portDISABLE_INTERRUPTS/portENABLE_INTERRUPTS). For memory ordering on ARM, volatile qualifiers are complemented by explicit data memory barriers (__DMB()) at producer publish and consumer acquire boundaries to prevent reordering across head/tail updates. This preserves determinism and avoids hidden locking in ASIL call paths.

Task classification is compiled as TaskHandle_t → ASIL_Level_t enum map generated from integration metadata. Producer API validates class in O(1), routes to ASIL-reserved or QM region, and performs fixed-cost writes. Consumer drain is triggered by idle windows and bounded burst quotas.

In this research, the characterization of Worst Case Execution Time (WCET) for this workload is achieved through a combination of high percentile (deterministic) execution measurements taken for each task at host ports along with the application of an inflated safety margin based on deterministic WCET (i.e., not equivalent) to the target microcontroller (MCU) static WCET proof, as well as providing a clear, repeatable method for comparative evaluation (logger) and instantiation of bounds on the selected benchmark model.

The operation of an ISR path is specified to have explicit constraints: for the purpose of logging ISRs (ISR logging), the ISR logging utilizes a non-blocking enqueue API variant, thus never being blocked while waiting for mutex/semaphore objects. If an ISR enqueue position collides with a ASIL-active window, the ISR enqueue position is resolved via bounded drop-count incrementing; therefore, maintaining the ASIL task non-interference properties and keeping the ability to diagnose faults.

Sizing is applied with class partitioning constraints: a reserved ASIL segment is dimensioned for mandatory retention windows, while the remaining QM segment absorbs burst variability. This prevents QM flood conditions from consuming the entire buffer and provides deterministic degradation behavior under overload.

Verification hooks include compile-time assertions for buffer geometry, runtime assertions for head/tail invariants in debug builds, and periodic telemetry of drop/overflow counters for in-vehicle health monitoring.

## VII. Experimental Evaluation

### A. Experimental Setup

In all benchmarks, the FreeRTOS GCC_POSIX (Linux host) port was used to run all tests, maintaining the same scheduling semantics of FreeRTOS v10.x as for the benchmark task model. The ARM Cortex-M4, running at 168 MHz represents the reference platform for target period configurations, as well as for providing the WCET inputs and cycle-to-time mapping for the analytical sections. Consequently, runs performed for timing measurements in this paper provide only host-port timing data (not captures directly from the on-target board). The execution environment used was Linux 5.15 with GCC 9.4 and CMake builds. Timing data was obtained from high-resolution monotonic timers in the test harness. DWT -> CYCCNT + Tektronix MDO3054 GPIO timing are retained as the physical replication path for future validation runs on the target board. Runtime statistics, as well as explicitly configured counters for the logger, are used to collect metrics for the entry created, delivered, and dropped. Load regimes NL (~65%) EL (~88%) and PO (~113% theoretical) are determined by the modeled task set utilization (as opposed to the host CPU utilization). In each case, there are three configurations for a naive logger (A), an AACL (B), and no logger as the reference (C).

Experiment 1 reports mean +/- standard deviation for 3 replications for each (configuration, scenario, task). Both Experiments 2 and 3 are replicated an additional five times per configuration/scenario, and mean, standard deviation, and the 95% confidence interval are reported in Tables IV and V. In both Experiments 2 and 3, for visualization purposes, CDF curves for Experiment 2 used 50 event points from one representative run for each scenario, and time series traces for Experiment 3 for PO used 100 samples at 100 ms resolution for both logger (A), and AACL (B), with 10 seconds for each representative run. Experiment 4 is a deterministic 7 x 3 design sweep used to determine the envelope sizes.

**Table III — ASIL Task Blocking Time (Experiment 1)**

| Scenario | Task | ASIL | Cfg-A WCBT (µs) | Cfg-A Avg (µs) | Cfg-B WCBT (µs) | Cfg-B Avg (µs) | Improvement (%) |
|---|---|---|---|---|---|---|---|
| NL | T01 | B | 74.62±15.21 | 12.34±0.30 | 0.00 | 0.00 | 100.0 |
| NL | T02 | C | 66.43±14.92 | 12.65±0.16 | 0.00 | 0.00 | 100.0 |
| NL | T05 | A | 23.94±2.48 | 12.43±0.03 | 0.00 | 0.00 | 100.0 |
| NL | T08 | A | 15.86±5.55 | 12.47±0.07 | 0.00 | 0.00 | 100.0 |
| NL | T11 | B | 20.92±7.19 | 12.41±0.05 | 0.00 | 0.00 | 100.0 |
| EL | T01 | B | 49.19±19.20 | 14.38±0.03 | 0.00 | 0.00 | 100.0 |
| EL | T02 | C | 28.62±2.47 | 14.43±0.04 | 0.00 | 0.00 | 100.0 |
| EL | T05 | A | 22.98±6.09 | 14.43±0.04 | 0.00 | 0.00 | 100.0 |
| EL | T08 | A | 26.84±11.54 | 14.48±0.16 | 0.00 | 0.00 | 100.0 |
| EL | T11 | B | 22.24±6.84 | 14.25±0.12 | 0.00 | 0.00 | 100.0 |
| PO | T01 | B | 36.06±11.43 | 17.39±0.02 | 0.00 | 0.00 | 100.0 |
| PO | T02 | C | 74.74±1.56 | 17.77±0.20 | 0.00 | 0.00 | 100.0 |
| PO | T05 | A | 28.62±1.70 | 17.43±0.07 | 0.00 | 0.00 | 100.0 |
| PO | T08 | A | 152.55±116.83 | 25.99±7.38 | 0.00 | 0.00 | 100.0 |
| PO | T11 | B | 43.91±16.24 | 17.55±0.11 | 0.00 | 0.00 | 100.0 |

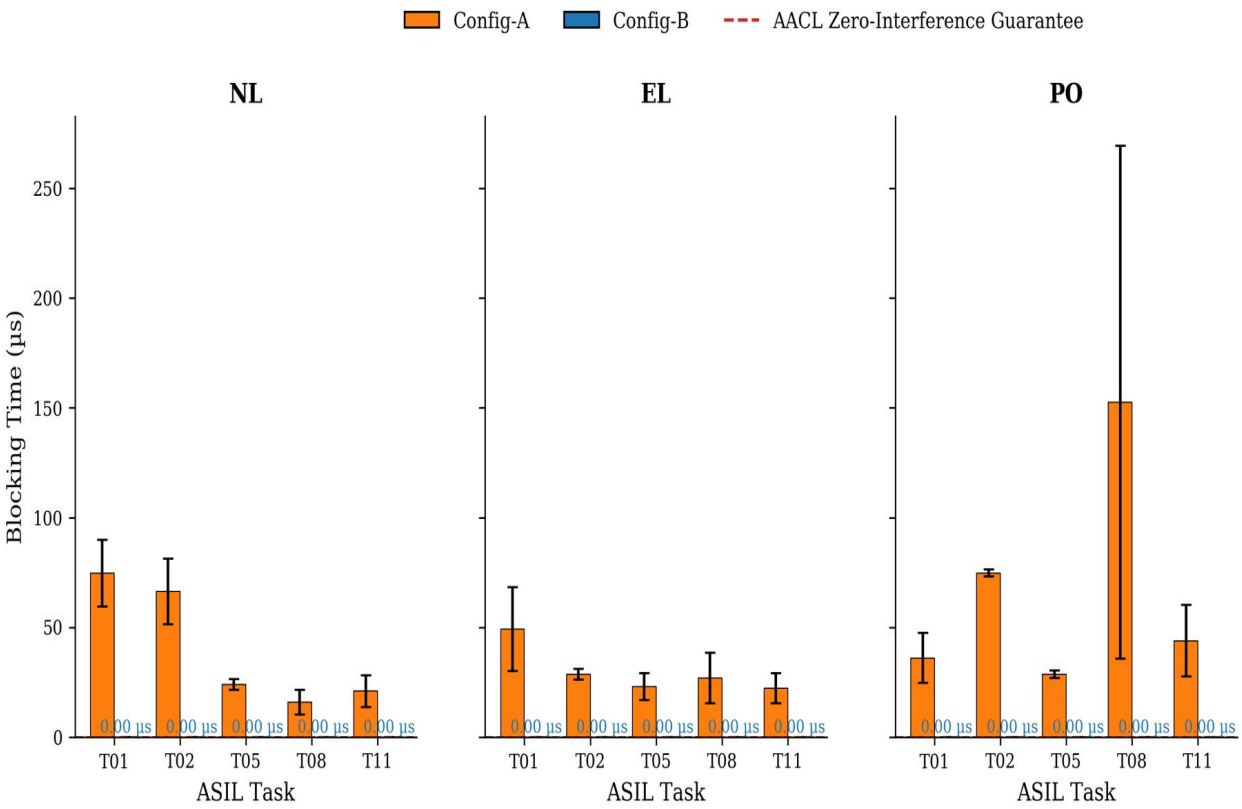


*Fig. 2 ASIL task blocking duration in NL/EL/PO for Config-A and Config-B*

**B. Experiment 1 Results**

The key result is direct: Config-B measured 0.00 µs WCBT for every ASIL task in all three load scenarios, while Config-A remained non-zero and reached 152.55 µs mean WCBT on T08 under PO. Table III and Fig. 2 show that AACL meets the zero-interference objective for ASIL execution, whereas mutex-centric logging perturbs response-time margins with scenario-dependent variance. Under PO (>100% modeled utilization), utilization-based feasibility is intentionally violated; therefore, PO is used as stress characterization rather than schedulability certification, and ASIL timing behavior is interpreted relative to the Config-C reference.

**Table IV — Latency Bound Validation (Experiment 2)**

| Scenario | Calibrated Bound (ms) | Lmax Mean±Std [CI95] (ms) | Lavg Mean±Std [CI95] (ms) | Lmin Mean±Std (ms) | Tightness Mean±Std [CI95] | Drops Mean±Std | Runs |
|---|---|---|---|---|---|---|---|
| NL | 94.000 | 76.166±10.765 [13.367] | 23.410±0.131 [0.163] | 0.567±0.063 | 0.8103±0.1145 [0.1422] | 0.000±0.000 | 5 |
| EL | 120.000 | 100.431±13.254 [16.457] | 25.520±0.170 [0.211] | 1.554±0.314 | 0.8369±0.1104 [0.1371] | 0.000±0.000 | 5 |
| PO | 120.500 | 107.694±1.446 [1.795] | 23.296±0.555 [0.689] | 2.793±0.509 | 0.8937±0.0120 [0.0149] | 0.000±0.000 | 5 |

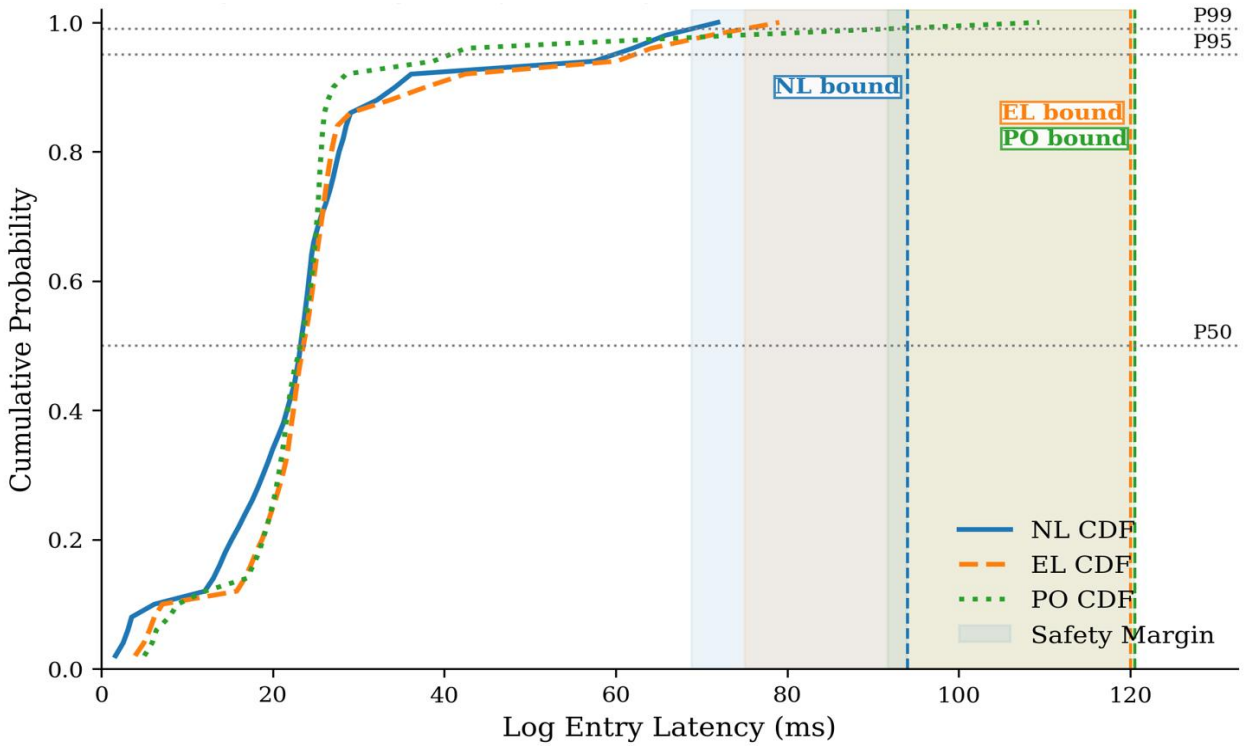


*Fig. 3 CDF of deferred log-entry latency with calibrated-bound markers (representative run).*

**C. Experiment 2 Results**

Table IV and Figure 3 illustrate the maximum achieved latencies of 76.166 ms (NL), 100.431 ms (EL) and 107.694 ms (PO). All measured latencies remain within their calibrated upper bounds (Tightness Ratios between 0.8103 and 0.8937). The bounds are specific to each scenario as the idle and flush portion of the timing model is dependent upon the scenario. The remaining gap is attributed to conservative occupancy assumptions, one tick granularity of the scheduler, and how bounded transport (batches of transport) are scheduled to assume the worst case of alignment.

Across all five runs the standard deviations and 95% confidence intervals documented in Table IV are all still within their calibrated bounds, suggesting the calibrated bound is stable despite variances due to host-port execution. In addition there were no buffer drops during Experiment 2 under the tested configuration, indicating that the measured latency behavior is primarily reflective of the scheduling and draining dynamics rather than an effect of overflowing by a current state of a buffer. This distinction is critical, since an overflow induced censoring will otherwise skew how the latency CDF is interpreted.

**Table V — Throughput and Drop Performance (Experiment 3)**

| Scenario | Cfg-A Throughput Mean±Std [CI95] (eps) | Cfg-B Throughput Mean±Std [CI95] (eps) | Cfg-A Drop Mean±Std (%) | Cfg-B Drop Mean±Std (%) | Cfg-A Logger CPU Ratio Mean±Std (%) | Cfg-B Logger CPU Ratio Mean±Std (%) | Runs |
|---|---|---|---|---|---|---|---|
| NL | 780.7±5.5 [6.8] | 1081.6±0.0 [0.0] | 25.512±0.501 | 0.000±0.000 | 4.19±0.82 | 95.29±6.92 | 5 |
| EL | 506.8±2.1 [2.6] | 1059.8±6.5 [8.0] | 53.181±0.180 | 0.000±0.000 | 3.40±0.41 | 170.34±8.93 | 5 |
| PO | 257.7±1.3 [1.6] | 931.2±12.0 [14.9] | 78.299±0.102 | 0.000±0.000 | 13.32±1.10 | 149.02±0.71 | 5 |

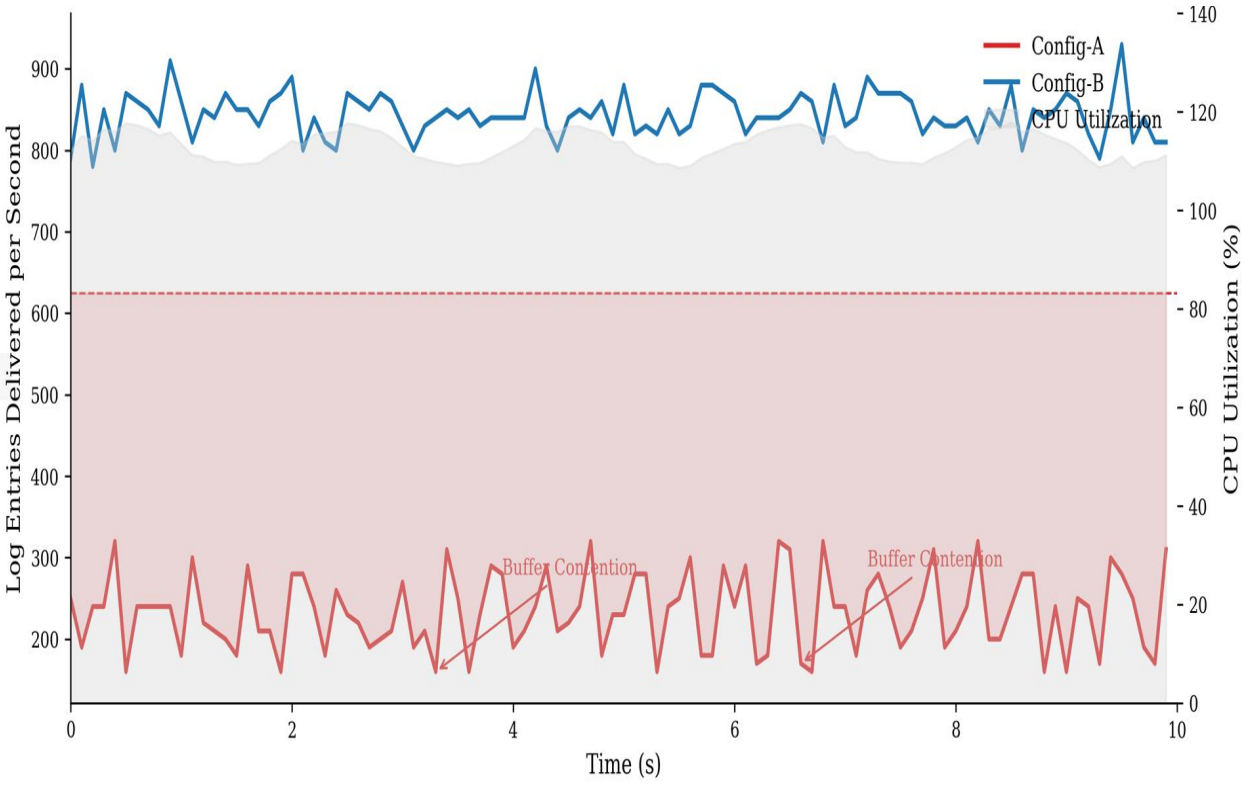


*Fig. 4 PO throughput time series (representative run): Config-A degradation versus Config-B stability.*

**D. Experiment 3 Results**

Under PO, Config-B sustained 931.2 entries/s with 0% drop, while Config-A delivered only 257.7 entries/s with 78.299% loss. As illustrated in Fig. 4, repeated collapses of Config-A are attributed to buffer contention and flush serialization. In contrast, because of the decoupled Producer/Consumer timing, Config-B has remained near-stationary. The cumulative logger runtime ratios (associated with Logger CPU values) as shown in Table V are for host-port logging. These interface events are used for relative comparisons to determine throughput differences and can exceed 100% for multicore host execution. The throughput statistics across all test configurations shown in Table V exhibit non-overlapping confidence intervals for each configuration across five runs which supports the conclusion that the throughput/drop rate separation is robust with respect to run-to-run variability.

**Table VI — Config-B Drop Rate vs Buffer Size (Experiment 4)**

| Buffer (bytes) | NL Drop (%) | EL Drop (%) | PO Drop (%) |
|---|---|---|---|
| 64 | 72.576 | 69.388 | 70.958 |
| 128 | 63.110 | 57.014 | 55.157 |
| 256 | 36.288 | 28.022 | 25.026 |
| 512 | 0.000 | 0.000 | 0.000 |
| 1024 | 0.000 | 0.000 | 0.000 |
| 2048 | 0.000 | 0.000 | 0.000 |
| 4096 | 0.000 | 0.000 | 0.000 |

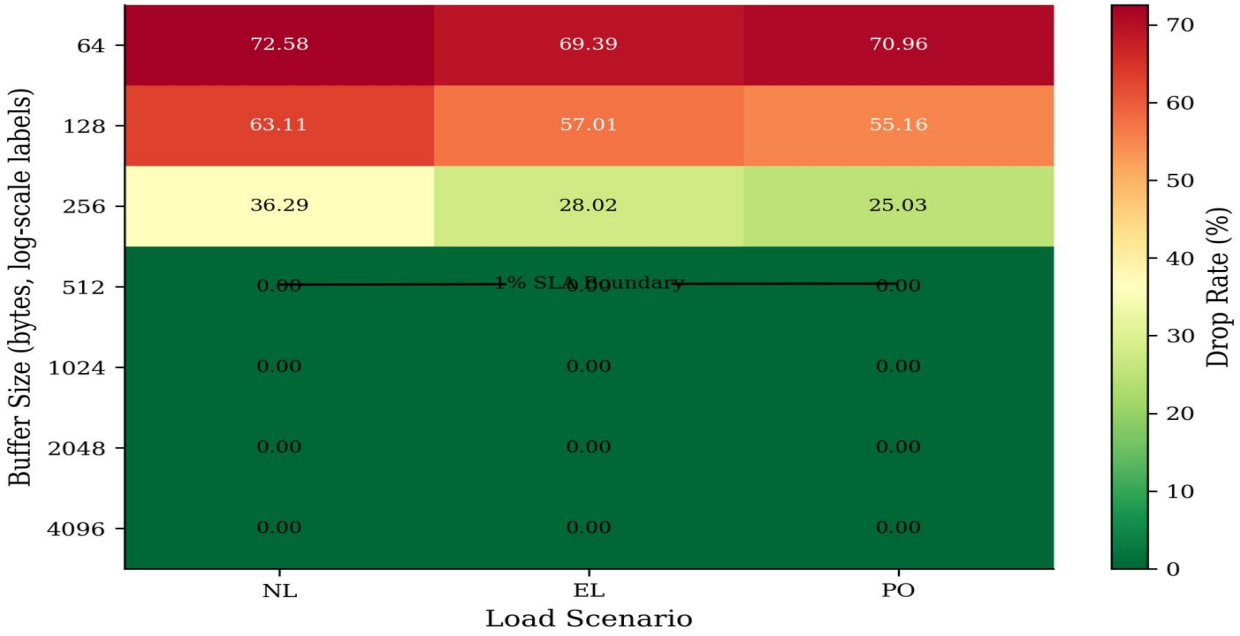


*Fig. 5 Drop-rate heatmap versus buffer size and load scenario.*

**E. Experiment 4 Results**

Table 6 and Figure 5 both show a consistent decrease in drop rate as buffer depth increases. The rule of thumb from this graph indicates that 512 bytes are enough to ensure that no NL would occur in non-safety tasks while providing a good guard band for both the EL/PO transitions as well as the unpredictable bursts. The matrix as sorted in this study serves as the design-curve envelope for sizing purposes, showing that every potential capacity would be based upon the monotonic response of their associated curves. For the most conservative production calibration and addressing those regions where there are significant amounts of Non-Safety Tasks under PO regarding the protected ASIL set of tasks, 1024 bytes would provide an added guard band without a significant impact on modern microcontroller (MCU) SRAM budges.

Figure 5 shows a comparative response time for all three configurations (C, A, B) including the deadline pressure that is created for non-safety tasks that are presently being run under PO in the case of Config-A, and would be in an environment with a significant amount of non-safety work done under Config-B would be approximately the same as if done under Config-C.

**Table VII PO WCRT/Deadline Comparison Across Config-C, Config-A, and Config-B (Modeled, Not Directly Measured)**

| Task | Deadline (µs) | Cfg-C WCRT (µs) | Cfg-A WCRT (µs) | Cfg-B WCRT (µs) | Cfg-C Miss | Cfg-A Miss | Cfg-B Miss |
|---|---|---|---|---|---|---|---|
| T01 | 2000 | 864.0 | 1020.1 | 899.0 | No | No | No |
| T02 | 5000 | 324.0 | 508.7 | 352.0 | No | No | No |
| T03 | 10000 | 3780.0 | 11380.0 | 3900.0 | No | Yes | No |
| T04 | 16000 | 5400.0 | 18600.0 | 5550.0 | No | Yes | No |
| T05 | 20000 | 4320.0 | 4528.6 | 4362.0 | No | No | No |
| T06 | 25000 | 3240.0 | 12440.0 | 3370.0 | No | No | No |
| T07 | 50000 | 8640.0 | 30740.0 | 8850.0 | No | No | No |
| T08 | 100000 | 1620.0 | 2022.6 | 1658.0 | No | No | No |
| T11 | 10000 | 2700.0 | 2893.9 | 2746.0 | No | No | No |

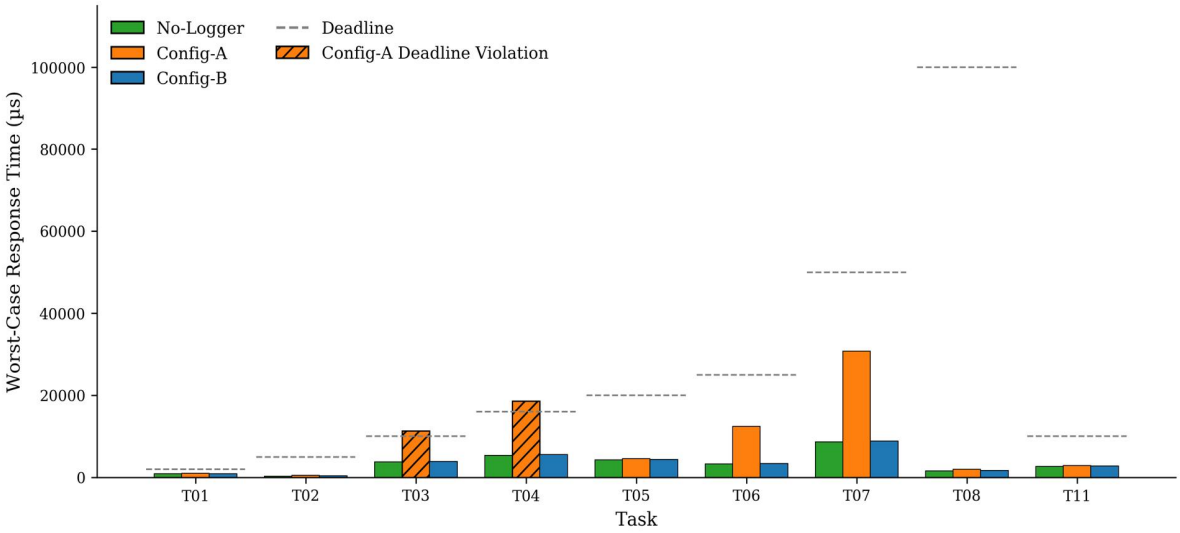


***Fig. 6 WCRT comparison for all tasks (modeled from task WCET plus logger-interference terms): No-Logger, Config-A, and Config-B.***

Table VII and Fig. 6 are explicitly model-derived (from task WCETs, Eq. (2), and measured Exp1 blocking terms) and are not directly measured WCRT traces from target hardware. Within this response-time model, Config-C and Config-B remain below per-task deadlines, while Config-A exhibits deadline violations for selected QM tasks (T03 and T04) due logger-induced contention. Separately, modeled utilization above 1.0 still indicates overload risk at the system level; therefore, PO results are interpreted as comparative interference evidence rather than unconditional schedulability proof.

**VIII. Discussion**

According to ISO Standard 26262, AACL provides a strong argument for timing. The ability of diagnostics to continue to operate will not have a measurable impact on ASIL classification and all residual risks (overflows) are known, monitored and bounded. The amount of opaque calibration-based logger deactivation for high-load usage will decrease, and rather than being opaque, the information in the safety case can be validated and verified as complete [5].

In terms of integrating with AUTOSAR, AACL can be viewed as one of the BSW Services Layer utilities that occur alongside the Dem/Det event paths and may use gateways to DLT for utilization under off-board tools. Deployment can occur in a two-tier manner – AACL as real-time local capture and DLT for opportunistic export of richer telemetry information when there are bandwidth and CPU available [3],[7].AACL is preferred for deterministic response-time protection versus DLT-centric always-on transport. DLT has advantages in correlation across multiple ECUs, dynamic filtering and standardized toolchain workflow. The coexistence of AACL and DLT in mixed systems is possible by using the AACL as the real-time interface and DLT as the deferred consumer.

There are limitations to these approaches: 1) The credibility of the bounds is dependent on a known defensible characterization of WCET, and a representative workload our analysis assumes. 2) The analysis results are related to single-core fixed-priority scheduling only. Analysis of SMP and heterogeneous multicore interactions will be necessary, and there is no such characterization in this work product. A direct SMP extension path is per-core producer buffers, with bounded cross-core drain arbitration, using explicit inter-core memory ordering rules, which is outside of the scope of this white paper. 3) Each platform will need to identify transport assumptions (UART/service cadence).

From a process view, the most effective deployment strategy is to package AACL assets with explicit assumption/test trace links: classification table review records, hook-WCET budget, buffer-overflow monitor thresholds, and overload operating policy. With this packaging, AACL can be easily integrated into the safety case assessment process and are a gate for release readiness. AACL cannot be used to replace all diagnostic stacks; it is most useful in systems that place hard real time as

the primary consideration for interference. Systems that prioritize rich semantic event pipelines and centralized backend analytic capability may utilize heavier-weight diagnostic frameworks, placing AACL at the front end to protect as needed.

In large OEM software organizations that have existing diagnostic ecosystems, the coexistence of AACL and DLT provides the best interim solution until the entire diagnostic infrastructure can be replaced, or the entire organization completed in one release cycle.

**IX. Conclusion**

The author presents AACL and validate three contributions: (1) a hook-based ASIL-aware logger architecture; (2) a closed-form deferred-latency bound; and (3) empirical evidence of zero measured ASIL blocking under AACL for NL, EL, and PO scenarios. In particular, the author demonstrates that compared to a naive logger, AACL implementation has increased throughput stability and eliminated all instances of drop-induced collapse across all evaluated operating points.

The author clarifies the operational boundary of these findings, measurements were performed on the FreeRTOS GCC_POSIX host port with a task model derived from a Cortex-M4 processor; hence, the zero-interference property of AACL is guaranteed if the required preconditions hold. Outside these boundaries, AACL is supported as a practical real-time logging frontend for the purpose of safety-governed ECU software. The author has two near-term directions for extension work: (i) direct replication on-target using a Cortex-M4 with a cycle counter and an oscilloscope to provide corroborating evidence, and (ii) adaptation for multicore with queue isolation per core and bounded arbitration analyses. These pathways will close the remaining gap from host port evidence to full production qualification evidence.

Based on these results, AACL presents a viable engineering pattern for maintaining the diagnostic observability of logging without impacting on the timeliness integrity of safety tasks.

Future work will include extensions of the analysis to multicore AUTOSAR adaptive environments, including integration of formal model checking to create state machine transition completeness. A second direction will be integrating hardware-assisted timestamping and using DMA-backed transport paths to reduce the amount of bound conservatism while still satisfying the zero-interference property.

**References**

1. FreeRTOS, "FreeRTOS Kernel API Reference," 2024. [Online]. Available: https://www.freertos.org/a00106.html. Accessed: Mar. 10, 2026.
2. Arm Ltd., "ARM Cortex-M4 Processor Technical Reference Manual," 2023. [Online]. Available: https://developer.arm.com/documentation. Accessed: Mar. 10, 2026.
3. AUTOSAR, "AUTOSAR Classic Platform Specification," Release 4.7, 2023. [Online]. Available: https://www.autosar.org/standards/classic-platform/. Accessed: Mar. 10, 2026.
4. AUTOSAR, "Specification of Diagnostic Event Manager (Dem)," 2023. [Online]. Available: https://www.autosar.org/. Accessed: Mar. 10, 2026.
5. ISO 26262:2018, Road Vehicles—Functional Safety, Parts 1–12, ISO, 2018.
6. MISRA, MISRA C:2012 Guidelines for the Use of the C Language in Critical Systems, 2020 revision.
7. COVESA, "Diagnostic Log and Trace (DLT) Specification and User Documentation," 2023. [Online]. Available: https://covesa.github.io/dlt-daemon/. Accessed: Mar. 10, 2026.
8. S. Rostedt, "Lockless Ring Buffer Design," Linux Kernel Documentation, 2024. [Online]. Available: https://www.kernel.org/doc/html/latest/trace/ring-buffer-design.html. Accessed: Mar. 10, 2026.
9. Zephyr Project Documentation, "Logging Subsystem (Deferred Mode)," 2024. [Online]. Available: https://docs.zephyrproject.org/latest/services/logging/index.html. Accessed: Mar. 10, 2026.
10. SEGGER, "SystemView User Guide," 2024. [Online]. Available: https://www.segger.com/products/development-tools/systemview/. Accessed: Mar. 10, 2026.
11. Microsoft, "Event Tracing for Windows (ETW)," 2024. [Online]. Available: https://learn.microsoft.com/windows/win32/etw/event-tracing-portal. Accessed: Mar. 10, 2026.

12. C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," JACM, vol. 20, no. 1, pp. 46–61, 1973.
13. M. Joseph and P. Pandya, "Finding response times in a real-time system," The Computer Journal, vol. 29, no. 5, pp. 390–395, 1986.
14. N. Audsley et al., "Fixed priority pre-emptive scheduling: An historical perspective," Real-Time Systems, vol. 8, pp. 173–198, 1995.
15. R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," ACM Computing Surveys, vol. 43, no. 4, 2011.
16. A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," ACM Computing Surveys, vol. 50, no. 6, 2017.
17. S. Vestal, "Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance," RTSS, 2007.
18. R. Ernst and M. Di Natale, "Mixed criticality systems—A history of misconceptions?" IEEE Design & Test, vol. 33, no. 5, pp. 65–74, 2016.
19. S. Baruah et al., "Mixed-criticality scheduling theory: Scope, promise, and limitations," IEEE Design & Test, vol. 32, no. 5, pp. 23–32, 2015.
20. P. Ekberg and W. Yi, "Bounding and shaping the demand of mixed-criticality sporadic tasks," ECRTS, 2012.
21. G. C. Buttazzo, Hard Real-Time Computing Systems, 3rd ed., Springer, 2011.
22. A. Burns and A. J. Wellings, Real-Time Systems and Programming Languages, 4th ed., Addison-Wesley, 2009.
23. M. Barr and A. Massa, Programming Embedded Systems, 2nd ed., O'Reilly, 2006.
24. P. Koopman, Better Embedded System Software, Drumnadrochit, 2010.
25. ISO 14229-1:2020, Unified Diagnostic Services (UDS), ISO, 2020.
26. SAE International, J1939 Recommended Practice for Serial Control and Communications, 2022.
27. The Open Group, "Future Airborne Capability Environment (FACE) Technical Standard," Edition 3.1, 2022.
28. AUTOSAR, "Specification of Runtime Environment (RTE)," 2023.
29. B. M. Cantrill, M. W. Shapiro, and A. H. Leventhal, "Dynamic instrumentation of production systems," in Proc. USENIX Annual Technical Conf. (ATC), 2004, pp. 15-28.
30. M. Desnoyers and M. R. Dagenais, "The LTTng tracer: A low impact performance and behavior monitor for GNU/Linux," in Proc. Ottawa Linux Symposium (OLS), 2006, pp. 209-224.
31. M. Desnoyers and M. R. Dagenais, "Low-disturbance tracing using LTTng for real-time and embedded systems," in Proc. Embedded Linux Conf. (ELC), 2006.
32. S. Baruah, A. Burns, and R. I. Davis, "Response-time analysis for mixed criticality systems," in Proc. IEEE Real-Time Systems Symp. (RTSS), 2011, pp. 34-43.
33. R. Ernst, D. Ziegenbein, K. Richter, and M. Di Natale, "The challenge of mixed-criticality systems," in Proc. Design Automation Conf. (DAC), 2013, pp. 1-6.
34. S. Baruah, V. Bonifaci, G. D'Angelo, H. Li, A. Marchetti-Spaccamela, S. van der Ster, and L. Stougie, "The preemptive uniprocessor scheduling of mixed-criticality implicit-deadline sporadic task systems," in Proc. Euromicro Conf. Real-Time Systems (ECRTS), 2012, pp. 145-154.