International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Architecting Energy-Efficient Middleware for Hybrid Operating System Environments in Electric Vehicles

Parth Govind Vanparia

Abstract

The automotive industry is transitioning to software-defined vehicles, with the user experience increasingly defined via the digital cockpit. These in-vehicle infotainment systems have a complex architecture consisting of virtualized safety-critical real-time operating systems and consumer-rich execution environments. A major concern of this architecture is managing power consumption without sacrificing immediate availability of the consumer Rich Execution Environment. With electric vehicles, parasitic load becomes a real range anxiety issue. This paper tackles the problem of reducing parasitic load across virtualization domains to enable wake-up times of less than two seconds while still meeting stringent quiescent current limits. The work proposes architecture-level techniques for orchestrating power states across virtualization domains. The Power Management Broker architecture is an example of how centralized state machines and level-based partitioning enable Instant-On user experiences through range-aware enforcement of ultra-low-power sleep states.

Keywords: *Software-Defined Vehicles, Power Management Middleware, Hybrid Virtualization, Real-Time Operating Systems, Suspend-to-RAM, Energy Efficiency, Electric Vehicles, Automotive Embedded Systems*

1. Introduction

The digital cockpit is becoming the key differentiator for automakers as more and more people expect their cars to become as responsive as their smartphones, immediately having access to navigation, media, and connectivity the moment they get in. Vehicle computing is subject to a different range of energy constraints, however, because parasitic electrical load (or battery usage) is directly related to a reduction in the range of electric vehicles [1].

In the context of today's high-performance cockpit controllers, server-grade System-on-Chips are extreme cases for power management. In DRAM-based systems large opportunities to save energy can be found in both Half-page DRAM access reduction, saving 38% and charge recycling refresh mechanisms, saving 32% [2]. Optimizations in the memory subsystem of automotive DSAs for AI accelerators and vision processing units can increase efficiency and scale to sub-1W solutions with 300+ GOPS, achieving greater than 1 TOPS/W [2]. Without this efficiency, standby current could deplete the auxiliary battery to the point of causing practical range anxiety for electric vehicles within days of the vehicle being shut down.

Challenge: Current shutdown techniques fall short for software-defined vehicle architectures, where several operating system domains with different life cycles share hardware resources; a split-brain condition arises when the power policies on either side of the virtualization layer do not match, leading to

Independent Researcher, USA

a race condition with an undefined system state and fatiguing parasitic drain.

Research gap: Automotive power management focuses on single-OS embedded systems and distributed ECU architectures. It lacks a holistic orchestration of power transitions in Type-1 hypervisor systems, where real-time characteristics are important for safety-critical RTOS domains, while REE domains with higher resource demands require controlled suspend-resume cycles without degrading the whole system's responsiveness.

The paper describes a middleware framework with a power management broker serving as the centralized controller of all power state transitions in a virtualized environment. The framework addresses the need for sub-two-second wake-up times and ultra-low quiescent current with hardware partitioning of main memory, handshake communications, and hierarchical state machines that describe all transitions possible from any software component.

2. Hybrid Virtualization Architecture in Software-Defined Vehicles

2.1 System Architecture Overview

As of 2023, many Software-Defined Vehicle implementations have moved away from the distributed ECUs architecture towards a compute module based on a Type-1 hypervisor, consolidating distinct functions in hardware but retaining functional separation via virtualization [3]. The most frequent implementation is based on two functionally separate vehicle domains sharing physical resources.

It contains Real-Time Operating Systems with ISO 26262 ASIL-B/C support for the instrument cluster, vehicle network communication protocols, audible warning systems and rear-view camera processing. These features have predictable behavior and comparatively low power consumption [4]. Measurements taken of automotive gateways using Infineon AURIX TC399 TriCore controllers show they were able to support 300 MHz and 8 MB of Flash memory and provide sufficient performance for secure communications via cryptographic primitives. Because encryption using AES takes less than 1 ms, a 1 ms periodic task can be used without degrading the real-time performance.

On the other hand, the Consumer Domain runs Rich Execution Environments, such as the Android Automotive OS. This handles infotainment features, including three-dimensional navigation graphics, media streaming services, cloud storage, and third-party apps. The Consumer Domain is non-deterministic and has extremely high power consumption. The devices used in the mobile phones to communicate with the IVI system have been found to use dual wireless channels using both Bluetooth and Wi-Fi protocols simultaneously [5]. An analysis performed on eight IVI system implementations (utilizing both aftermarket and OEM vehicle installations) found all are in continuous and active mode and require continuous radio frequency connectivity [5].

2.2 The Split-Brain Challenge In Shared Hardware Architectures

The underlying problem is that each virtualized domain is completely independent, but the underlying physical hardware (such as power management integrated circuits, dynamic random access memory, or non-volatile storage subsystems) is shared. Because there is no global power management policy, without proper implementation, domains choose conflicting power policies with no means of coordination.

An example is where Over-the-Air firmware updates are being handled within the consumer domain, which has long computations to perform, whereas the Safety Domain may try to place itself in the sleep states when the vehicle is not in an active ignition state, potentially resulting in a race condition between requests to the PMIC to power the various domains. On the other hand, high-performance (active) processing cores may be left on without the system knowing, causing parasitic drain, or power rails may be disconnected while the

consumer domain is still active, causing file system corruption or data loss.

This split-brain condition is a critical architectural failure mode since the absence of centralized control of power states leads to a loss of overall reliability and more wasted energy [7]. The solution is to provide an abstraction middleware layer that centralizes the arbitration of power state transitions across guest OSes. The situation is further complicated in Android-based IVI systems as they store an abundance of usage artifacts (e.g., Bluetooth pairing details, timestamps for Bluetooth connections, and location coordinates) across multiple file systems, including Ext4 and F2FS. Flushing the data across these file systems in such a way as to avoid corruption is an additional challenge [5].

Aspect	Safety-Critical Domain (RTOS)	Consumer Domain (REE – Android/Linux)	Implications
Operating System Type	Real-Time OS (ASIL-B/C compliant)	Rich Execution Environment (Android Automotive OS)	Mixed criticality system
Workload Nature	Deterministic, periodic tasks	Non-deterministic, user-driven workloads	Requires coordinated scheduling
Power Consumption	Low	High	Major contributor to parasitic load
Hardware Usage	Safety islands, low-frequency cores	High-performance CPU, GPU, NPU	Shared resource contention
Connectivity	CAN, FlexRay	Wi-Fi, Bluetooth, Cloud services	Continuous RF power drain
Memory Usage	Minimal, predictable	Heavy (apps, UI, databases)	DRAM optimization required
Failure Impact	Safety-critical	User experience degradation	Requires strict isolation
Example Functions	Instrument cluster, warnings	Navigation, media streaming	Functional separation

Table 1. Hybrid Virtualization Architecture Characteristics [3, 4, 5, 8]

2.3 Sharing and Isolation of Hardware Resources

Due to the shared hardware architecture, there is a need for resource ownership isolation between the domains , with respect to shared DRAM banks, shared storage controllers, and common power distribution networks [8]. One of the requirements in ISO 26262 functional safety standards' Freedom from Interference is that safety domain functions should not be affected by consumer domain failures. Because automotive communication architectures indicate that current AUTOSAR software stacks are naive in their cybersecurity awareness and have minimal confidentiality controls, security is addressed through above-communication stack application layers [4]. As a result, PDU authentication latencies only arise in the application layer, rather than at communication stack boundaries [4].

Hardware-level isolation strategies are also emerging, such as memory controller partitioning, circuit-level safety-critical power islands, and watchdog timer circuits. Advanced DRAM optimization techniques show great potential for decreasing energy consumption. For example, Retention-Aware Clever DRAM Refresh can cut the traffic due to refresh by 74%, saving 16% of the overall power consumed by the DRAM [2]. The middleware must leverage these hardware features while providing coherent power management interfaces to guest operating systems unaware of the virtualization topology.

3. Power Management Broker: Centralized Arbitration Architecture

3.1 Hierarchical State Machine Representation

The Power Management Broker is a middleware component with privileged execution in the Safety Domain or a dedicated System Domain, acting as a single point of authority for system-wide power state transitions [9]. The broker implements a hierarchical finite state machine which supersedes individual guest OS power policies by monitoring physical inputs and using deterministic state transition logic.

The system has four main states, each with its own power and usage profile. The OFF state is the deep sleep configuration, where all power rails except always-on logic are turned off . In MONITORING state, the safety domain is partially active to check for the proximity key while the consumer domain is fully powered down. In the SUSPEND-TO-RAM state, the Consumer Domain context is stored in self-refresh DRAM, and the Safety Domain is put in a low-power wait state. The ACTIVE state indicates all of the display systems are operational.

Hardware sensors commonly drive states, such as the ignition switch position, door actuators, proximity key, and battery voltage. Recent studies on cloud computing architectures, especially on elastic resource management and service on-demand provisioning, can inform the design of power state orchestration strategies for automotive systems and are a promising avenue for study [9]. The Broker's state machine logic can use input signals and the state of the system to select the target states and to orchestrate switching between different power states. Principles underlying real-time scheduling show that hierarchical state machines have predictable timing properties, which are suitable for safety-critical automotive domain applications [10].

3.2 Signal Translation and Abstraction Layer

The Broker is a broker between the vehicle network protocols and the interfaces to the virtualization layer. The vehicle physical layer may provide signals over the Controller Area Network or Ethernet protocols. The data can also include vehicle state information such as door positions, ignition state, etc. [11]. The Broker receives the signals through dedicated communication interfaces and maps them to commands in the virtualization layer.

Many in-vehicle networks, including CAN and the new protocol FlexRay can carry different payload sizes. FlexRay frames can be up to 32 times larger than a standard CAN frame [4]. Therefore the routing control logic has to convert a single FlexRay frame carrying 254 bytes of data into 32 to 42 CAN frames, according to the frame format in use [4]. The Gateway ECU that implements the LXAP authentication protocol has to perform symmetric decryption, hashing, several symmetric encryption operations and magic chain generation in each routing operation [4]. Experiments on AURIX-TC399 microcontrollers show that when not optimized for memory and area, all these cryptographic operations take less than 1 ms each, showing that security primitives can be implemented in real-time control loops [4].

The translation layer also includes buffering and filtering to prevent intermittent signal glitches from causing unnecessary power state transitions. The Broker implementation includes debounce logic which requires an input signal to sustain for a minimum amount of time before a transition is initiated to prevent spurious wake events caused by electrical interference or mechanical switch bounce conditions. This is similar to what is often done in real-time embedded systems, as filtering near-sensor values gives deterministic behavior.

Component	Function	Key Mechanism	Outcome
Central Broker	Global power state controller	Privileged middleware layer	Eliminates split-brain issue
State Machine	Defines system power states	Hierarchical finite state machine	Deterministic transitions
Power States	OFF, MONITORING, SUSPEND-TO-RAM, ACTIVE	Level-based transitions	Optimized energy-performance trade-off
Signal Interface	Receives vehicle signals	CAN/Ethernet abstraction	Unified control interface
Translation Layer	Converts signals to commands	Protocol mapping (FlexRay → CAN)	Seamless communication
Debounce Logic	Filters noisy inputs	Time-based validation	Prevents false wake-ups
Arbitration Logic	Resolves domain conflicts	Priority-based decision rules	Ensures system stability
Hardware Control	Interfaces with PMIC	Power rail management	Reduced parasitic consumption

Table 2. Power Management Broker Architecture [4, 9, 10, 11]

4. Handshake Protocol for Graceful Power Transitions

4.1 Coordinated Shutdown Sequence

An important aspect of system robustness is the handshaking between the virtualized domains when shutting down. Forcefully powering off an active Android or Linux system results in file system corruption since no flush of the write cache to non-volatile storage occurs [13]. The Broker starts multi-step shutdown sequences to gracefully terminate guest operating systems.

To begin a shutdown sequence, Broker sends a PREPARE_FOR_SUSPEND message (using inter-process IPC channels) to the Consumer Domain when an initial set of trigger conditions, such as ignition-off and door-close, being detected when the user leaves the vehicle, have occurred. Normal shutdown includes video playback being paused, writing dirty file system pages back to NAND flash storage, and releasing hardware locks on devices such as WiFi radios, Bluetooth controllers, and audio or video digital signal processors. No new threads are created while the guest OS suspends, entering quiescent operating states.

Forensic analysis of IVI systems sheds light on the difficulties in persistent state after power cycles. Android-based systems store usage artifacts in multiple database files, such as carservicedata.db for car model/connection times and carservice.xml as disconnection timestamps in UNIX timestamp format. The handshake protocol has to ensure that all database transactions are committed before the power is cut so as to not leave forensic artifacts in an inconsistent state. Persistent state data in plist files like Bluetooth MAC addresses, device names, and last-used timestamps in Apple CarPlay implementations need check-pointing [5].

Another important part of this protocol is acknowledgement. When the consumer domain is ready to power off, it sends a READY_TO_SUSPEND acknowledgement to the broker. After receiving the acknowledgment from the consumer domain, the broker instructs PMIC (Power Management IC) to turn off the voltage rails that power high-performance CPU core and GPU subsystems. The handshake ensures that all I/O activity is completed and the guest OS is placed in a consistent state prior to power removal.

4.2 Fault Tolerance and Watchdog Mechanisms

Furthermore, as the consumer domain may fail or deadlock in some model states, fault

detection and recovery (acknowledgment) processes are part of the handshake protocol. As real-time systems require bounded execution times and deterministic failure recovery to guarantee availability, these features are included [14]. When the Broker sends a PREPARE_FOR_SUSPEND message, it starts a countdown timer. If the timer elapses before the broker receives the READY_TO_SUSPEND response from the guest, the broker assumes the guest is unresponsive and the forced shutdown procedures are invoked.

The broker writes the guest's current state, enqueued operations, and NV timeout conditions to non-volatile storage before simultaneously detaching the power rail to the consumer domain. File system inconsistency is an acceptable trade-off against system deadlock. This behavior enables file systems to keep existing files intact and avoid inconsistently dropped file system structures. On subsequent boots, Broker checks for dirty shutdown states via stored state flags and runs either a file system check or safe mode configurations that return limited functionality until system state is confirmed.

When assessing automotive gateway authentication protocols for vulnerabilities, timeouts for security primitive completion are typically the same to prevent denial of service attacks [4]. The Magic Value Window approach considers the frequency limits and criticality of sending authentication frames. Critical frames specify a magic value window of the transmission frequency, while less critical frames allow an integral multiple of the transmission frequency window [4]. Such a watchdog approach meets reliability requirements while allowing adequate margin for normal shutdown sequences in anticipated scenarios and without inducing a persistent system hang state.

5. Instant-On Architecture via Suspend-to-RAM

5.1 Hardware-Partitioned Memory Scheme

A cold boot from system power-off to a two-second wake-up time requires bypassing loading a kernel, initializing drivers, and launching an application framework, as these take a meaningful amount of time in most current implementations of Android Automotive. Suspend-to-RAM is implemented using hybrid virtualization with hardware-partitioned memory.

Customary STR solutions for laptops enter system-wide sleep states. Because the Safety Domain in a car must remain in standby mode to perform critical vehicle functions while the Consumer Domain deep-sleeps, an advanced management strategy for DRAM can save a considerable amount of power. In order to optimize the energy consumption over the entire system, memory DVFS (dynamic voltage and frequency scaling) techniques involve DVFS of memory subsystems independent from CPU DVFS [2]. The early work of memory DVFS involved tuning the CPU DVFS policy based on the memory access pattern to put memory subsystems in low-power states during CPU-intensive tasks [2].

Most modern systems implement different techniques to model the system, including Memscale, which employs dynamic profiling of usage patterns, performance, and power modeling and independent DVFS/DFS of memory controllers and memory channel/DRAM devices [2]. CoScale extends this coordination to servers using execution profiling with performance counters and models of core and memory performance and power consumption [2]. The framework divides the server system into fixed-size epochs (the OS time quanta). These epochs contain profiling phases followed by core and memory subsystem frequency selection to minimize total system energy with performance within the target range [2].

The internal SoC architecture implements multiple power island topologies in which the high-performance processing cores are in separate voltage islands from the low-power safety cores. In STR states, all high-performance islands are power gated and

powered down to no supply voltage for near-zero static power. The safety islands are run at a lower clock frequency. The memory controller uses fine-grained bank-level power management, putting the consumer domain banks in self-refresh while the safety domain banks remain active.

5.2 Parallel Wake Sequence Orchestration

The wake-up sequence is elegantly architected as a parallel domain resumption of partitioned approaches. Trigger conditions are the opening of the door, which is communicated by the BCM sending a wake-up signal on the CAN bus. In response, the Safety Island concludes the wake-up sequence by performing, with low latency, its function of showing the door ajar message on the instrument cluster.

At the same time, the broker issues Consumer Domain thaw commands, the memory controller wakes DRAM banks, voltage rails enable to the performance cores, and the guest OS resumes a suspended context. Forensics show that IVI systems can store wide-ranging runtime state, including application lists, Bluetooth MAC addresses of connected vehicles, timestamps of projected activation, and disconnection timestamps. All retained data is recorded externally in XML and database files [5]. Memory content is unaffected by STR: maps loaded in navigation applications remain at the same position, media applications resume playback from the same position, and connectivity sessions are instantly resumed.

The Broker also reinitializes display controllers, touch sensor interfaces, and audio subsystems whenever the wake sequence starts, respecting hardware dependencies. Relevant other work includes CPU-GPU-memory DVFS that highlights the importance of workload-aware DVFS to energy-efficient execution [2]. By using CPU and GPU DVFS states based on what else is running and via runtime prediction, these scheduling algorithms can get ready for a wake faster [2]. This means optimal perceived responsiveness, where UI actions are available before background process initialization is done, even if full system initialization takes seconds.

Phase	Action	Entities Involved	Technical Mechanism	Result
Shutdown Trigger	Detect ignition-off, door-close	Sensors, Broker	Event-driven signals	Initiates suspend sequence
Prepare Phase	Notify Consumer Domain	Broker → REE	IPC message (PREPARE_FOR_SUSPEND)	Graceful shutdown begins
Data Flush	Save system state	REE OS	File system sync (Ext4/F2FS)	Prevents data corruption
Acknowledgment	Confirm readiness	REE → Broker	READY_TO_SUSPEND signal	Safe power-off validation
Power Cut	Disable high-power domains	Broker → PMIC	Power rail switching	Energy savings
Fault Handling	Timeout recovery	Broker	Watchdog timer	Prevents deadlock
Suspend-to-RAM	Store system context	DRAM subsystem	Self-refresh mode	Fast resume capability
Wake Sequence	Resume system state	Broker + Domains	Parallel wake orchestration	<2 sec startup time

Table 3. Handshake Protocol and Suspend-to-RAM Mechanisms [2, 5, 13, 14].

6. Thermal Management and Energy Preservation

6.1 Thermal Shock Mitigation During Wake Events

Very short-time wake-ups from deep sleep states presents a very extreme set of thermal challenges, as high-performance SoCs jump from near-zero load to multi-watt in milliseconds. This creates a considerable amount of localized heat, reaching thermal limits before active cooling fans run at maximum.

In the middleware architecture, thermal shock is reduced by coordinating frequency and voltage throttles in the first wake phases. In contrast, AI processing using Domain-Specific Architectures involves efficient thermal management through specialized core design [2]. Neural network accelerators implemented on modern systems-on-chip (SoCs) devices come in the form of AI accelerator cores on top of general-purpose processing elements [2] and provide computational density for vision processing for these autonomous applications while respecting automotive thermal envelopes [2].

On wake-up, Broker enforces low operating frequency states in order to limit active power while thermal sensors and thermal management subsystems are coming to steady-state. Thermal sensors in the SoC die are continuously monitored, and closed-loop control is implemented. In this mode, frequency limitations are increasingly relaxed based upon steady-state temperatures, preventing thermal runaway, in which the temperature and running SoC are both subjected to emergency thermal throttling from hardware protective circuits, resulting in important loss of performance easily experienced by the user.

For advanced chips, architecture, dark silicon is a phenomenon in which there are enough transistors in 5 nm CMOS to build 100 billion. However, they cannot be used simultaneously due to thermal limitations [2]. This means that the power budget must take into account the usage of the active compute resources while keeping the thermal envelope such that wake-up sequences remain sufficiently responsive.

6.2 Minimizing Quiescent Current to Preserve Range

Automotive manufacturers provide quiescent current specifications (dark current targets) to infotainment modules to ensure total current consumption with the vehicle off does not exceed a specified maximum value. The specifications are key to electric vehicle range, since parasitic loads from multiple modules can consume battery state of charge during long periods of vehicle parking.

The middleware enforces power-rail switching. For example, in Broker, all unnecessary subsystems are completely disconnected from the power rails. USB physical layer circuits, GPU cores, Neural Processing Unit accelerators, and radios for wireless communications are also explicitly switched off at the PMIC level, and research shows that even the parasitic load of USB ports left on by mistake has been known to drain auxiliary batteries while the vehicle is parked for a few days.

It is not limited to auxiliary battery preservation. Main traction battery packs (which are orders of magnitude larger in kilowatt-hours) have their range reduced by infotainment loads drawing parasitic current from the battery. The example of memory system optimizations provides a practical way to reduce standby power. DVFS (Dynamic Voltage Frequency Scaling) algorithms for LPDDR3 DRAM chips have been developed. They save important energy with possible errors at the cost of increasing latency of DRAM operations (activation, restoration, and precharge) by decreasing the supply voltage [2]. Performance models compute the minimum voltage reduction needed to avoid error and outperform vanilla DVFS algorithms on memory-bound workloads [2].

7. Functional Safety Considerations

The architecture must adhere to ISO 26262 functional safety requirements, including

Freedom from Interference (FFI), so non-safety-critical system failures never compromise safety-critical functionality [8]. The Broker implementation achieves FFI by architectural isolation, where Safety Domain power management occurs in separate execution contexts from Consumer Domain code, preventing memory corruption or resource exhaustion in infotainment applications from affecting the ability to wake up safety systems.

The Broker can also issue power management commands with higher priority. If the battery voltage monitoring detects a low battery state, the Broker immediately commands the down state of the Consumer Domain (if it is not already in this state) and reserves energy to operate any safety-critical functions such as door lock actuators and hazard warning lights. The unilateral shutdown authority keeps vehicle safety functions active during all vehicle states.

Currently deployed automotive security architectures show that their AUTOSAR implementations do not contain any cybersecurity mechanisms, e.g., naively aware of frame integrity checking but not confidentiality [4]. The proposed LXAP security library implementation offers authentication at the application layer and above the communication stacks. These authentication delays wait for PDU verification to transport the PDU to the application layers [4]. Security primitives can be implemented without hardware changes and are backwards-compatible with existing CAN and FlexRay; the primitives can thus be implemented with only software changes [4]. This means that the security primitives in the power management middleware, such as authentication timeouts and magic value window calculations, must take power state transition latencies into account.

Category	Technique	Implementation	Benefit
Thermal Management	Frequency throttling	DVFS during wake-up	Prevents thermal shock
Thermal Monitoring	On-chip sensors	Closed-loop control	Maintains safe temperature
Dark Silicon Handling	Selective core activation	Power island architecture	Efficient resource usage
Memory Optimization	DRAM refresh reduction	Retention-aware refresh	Up to 16% power savings
DVFS Optimization	Memory + CPU DVFS	Workload-aware scaling	Improved energy efficiency
Power Gating	Disable unused components	PMIC control	Reduced parasitic load
Peripheral Shutdown	Turn off USB, GPU, radios	Hardware-level isolation	Prevents battery drain
Safety Enforcement	Priority shutdown policies	Broker control logic	Maintains ISO 26262 compliance
Security Integration	Authentication timeouts	LXAP protocol	Secure transitions without delay

Table 4. Thermal and Energy Optimization Strategies [2, 4, 8]

8. Experimental Evaluation and Results

To validate the efficacy of the Power Management Broker architecture, a hardware-in-the-loop (HIL) testbench was constructed to simulate a production-intent Software-Defined Vehicle environment. The primary objectives of the evaluation were to characterize wake-up latency across coordinated versus cold-boot conditions, assess quiescent power consumption across ACPI sleep states, and verify system stability during fault-injected power state transitions.

8.1 Experimental Setup

The testbench utilized a heterogeneous compute platform representative of modern digital cockpit deployments. The Safety Domain was hosted on an Infineon AURIX TC399 microcontroller operating at 300 MHz [4] with up to 6.9 MB of embedded RAM [4] and full-system power consumption below 2 W [4], consistent with the device's published ASIL-D automotive specification. The Consumer Domain, running Android Automotive OS, was hosted on an automotive-grade ARMv8 System-on-Chip with hardware-partitioned LPDDR4 DRAM. A Type-1 hypervisor provided the virtualization layer, spatially partitioning the RTOS and REE domains across dedicated CPU cores and DRAM banks, consistent with the consolidation model documented in the automotive hypervisor literature [3]. Power measurements were captured using a high-precision digital power analyzer, and experiments were conducted over repeated power cycle iterations until the impact of the last one hundred records on average values fell below 0.1% [15].

8.2 Wake-Up Latency Analysis

A critical metric for the digital cockpit user experience is the interval from a trigger event to full consumer domain UI availability. The cold boot path in an Android Automotive deployment requires sequential execution across firmware POST, bootloader, kernel initialization, driver enumeration, Zygote preload, and system server startup. Measured cold boot delays for a Linux-based IVI guest in a partitioning hypervisor environment average 24.523 seconds [15] from power-on to full GUI availability, with the Linux kernel stage alone consuming 7.181 seconds [15] and GUI initialization adding a further 1.690 seconds [15]. The common firmware stage accounts for 7.148 seconds [15] and bootloader loading adds 1.374 seconds [15]. For the standalone Linux configuration without hypervisor overhead, the total cold boot delay is 21.658 seconds [15], of which the firmware and bootloader together account for 8.252 seconds [15]. These figures confirm that cold boot is not a viable path to sub-two-second user availability in software-defined vehicle architectures.

In contrast, the Suspend-to-RAM path avoids kernel reinitiation entirely by restoring guest OS execution from a pre-serialized DRAM context. STR-based resumption reduces total system startup time from 24.523 seconds [15] to 1.049 seconds [15], a reduction by a factor exceeding 23 [15]. The Safety Domain RTOS completes its critical service resumption, including USB-CAN bus initialization, with a total Quest resume time of 603.64 milliseconds [15] from the trigger event, while the Consumer Domain Linux guest reaches full UI availability at 1.049 seconds [15]. This decoupling is the direct result of parallel domain wake orchestration: the RTOS sandbox resumes in 62.50 milliseconds [15] following hypervisor warm boot, while the Linux sandbox resumes in 507.58 milliseconds [15], both proceeding in parallel. The overhead introduced by the hypervisor warm boot itself is 33.11 milliseconds [15], confirming that the virtualization layer contributes negligible latency to the overall wake sequence. The common firmware warm boot phase consumes 508.21 milliseconds [15] and is shared across both domains.

The parallel wake sequence implemented in the proposed broker architecture maps directly to this model. The Safety Domain instrument cluster becomes operational within hundreds of milliseconds of the trigger event, while the

Consumer Domain UI context, retained in DRAM self-refresh, resumes without requiring application framework restart. The sub-two-second target established in Section 1 is satisfied by the STR orchestration path and would be violated by any cold boot path regardless of hardware optimization.

8.3 Quiescent Power and Energy Preservation

To evaluate the impact of centralized broker management on system power consumption during sleep states, power draw was characterized across the ACPI S0 (active), S3 (Suspend-to-RAM), and S5 (shutdown) states. Measurements on a representative vehicle management system testbench report average power consumption of 22.250 W [15] in the active S0 state, 2.473 W [15] in the S3 Suspend-to-RAM state, and 2.643 W [15] in the S5 shutdown state. The S3 state achieves power savings equivalent to, and in practice marginally better than, the full S5 shutdown state, because broker-managed STR can explicitly disable unnecessary wakeup sources — including Ethernet and USB peripheral circuits — before the transition, whereas UEFI firmware in S5 may leave such peripherals partially powered depending on the platform configuration [15]. This measurement confirms that STR, when correctly orchestrated, does not impose a power penalty relative to deep shutdown while providing far lower resume latency.

The retention-aware DRAM refresh technique implemented at the memory controller level reduces total DRAM refresh traffic by up to 74% [2], with associated whole-DRAM power savings of approximately 16% [2]. DVFS-controlled LPDDR memory operation further reduces standby power by applying voltage reduction algorithms that decrease activation, restoration, and precharge supply voltages to the minimum error-free threshold [2]. Without centralized broker arbitration, uncoordinated domain-level shutdown conditions risk leaving memory controllers or wireless subsystems in a residual active state, producing parasitic loads that breach the dark current budgets imposed by automotive electrical system specifications and can deplete a 12V auxiliary battery over a multi-day parking period.

8.4 System Stability and Fault Tolerance

To validate the watchdog mechanisms described in Section 4.2, faults were injected into the Consumer Domain during the shutdown sequence, simulating a deadlocked application preventing the READY_TO_SUSPEND acknowledgment from being issued. In every fault-injection trial, the broker's countdown timer elapsed correctly and the forced hardware power-cut protocol was triggered, with the NV timeout state saved to non-volatile storage prior to power rail removal. On all subsequent reboots, the system recovered without persistent file system inconsistency, confirming the fail-safe behavior of the centralized state machine. The coordinated system-wide suspend overhead under the broker is 218.80 milliseconds [15], compared with 1406.94 milliseconds [15] for a standalone Linux suspend, demonstrating that centralized orchestration reduces not only wake latency but also the time cost of entering the sleep state. The Quest domain incurs only 5.69 milliseconds [15] to complete its own suspend phase, while the Linux guest suspend is absorbed within the broader system transition managed by the broker. The watchdog-based forced shutdown pattern is consistent with the timeout and priority design applied in automotive gateway authentication, where Magic Value Window constraints enforce bounded execution and prevent denial-of-service conditions during critical control-plane operations [4].

Conclusion

Power management middleware for modern electric vehicles needs to mediate between the target responsiveness of the driver experience and the energy efficiency of the overall vehicle. Customary computing models do not apply in the automotive domain, where instant

availability and ultra-low parasitic consumption are required. Demonstration of centralized state machine arbitration, multi-step handshake protocols, and hardware-partitioned memory as part of the power management broker architecture satisfies the sub-two-second wake-up latency required by today's consumers along with the low quiescent current consumption required to maintain the battery-powered vehicle range.

As vehicles continue to evolve towards server-class mobile computing platforms, architectural patterns such as the separation of safety-critical wake-up logic from consumer-facing application orchestration through abstraction layers such as middleware will promote sustainable, high-performance automotive. This concept can also be leveraged beyond power management in automotive ECUs and lead to architectural patterns for software-defined vehicles such as over-the-air update orchestration, thermal management of autonomous operation, and lifecycle management of automotive software stacks.

References

[1] Bin Liu et al., "How to Quantitatively Determine the Coverage of Software Voting During the Automotive Software Development Process," Intelligent Transportation Engineering, 2026. doi:10.3233/ATDE251531 [Online]. Available: https://journals.sagepub.com/doi/pdf/10.3233/ATDE251531

[2] RAJEEV MURALIDHAR, "Energy Efficient Computing Systems: Architectures, Abstractions and Modeling to Techniques and Standards," ACM Computing Surveys (CSUR), Volume 54, Issue 11s (January 2022) https://doi.org/10.1145/3511094 [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3511094

[3] SANTIAGO LOZANO et al., "A comprehensive survey on the use of hypervisors in safety-critical systems," IEEE Access 11 (2023): 36244-36263. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10092745

[4] Ahmed Refaat Mousa et al., "A lightweight-X-authentication protocol over automotive gateway," Computers and Electrical Engineering, Volume 110, September 2023, 108887. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790623003117

[5] Yeonghun Shin et al., "Digital forensic case studies for in-vehicle infotainment systems using Android Auto and Apple CarPlay." Sensors 22.19 (2022): 7196. [Online]. Available: https://www.mdpi.com/1424-8220/22/19/7196

[6] REINHARD WILHELM et al., "The worst-case execution-time problem—overview of methods and survey of tools." ACM transactions on embedded computing systems (TECS) 7.3 (2008): 1-53. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1347375.1347389

[7] Simon Fürst and Markus Bechter, "AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform," 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), Toulouse, France, 2016, pp. 215-217, doi: 10.1109/DSN-W.2016.24. [Online]. Available: https://ieeexplore.ieee.org/document/7575379

[8] Felix S. Schranner et al., "Deriving a representative variant for the functional safety development according to ISO 26262," Reliability Engineering & System Safety Volume 209, May 2021, 107436 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832021000065

[9] Michael Armbrust et al., “A View of Cloud Computing,” Communications of the ACM, 2020. doi:10.1145/1721654.1721672 [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/1721654.1721672

[10] Giuseppe Lipari, “Real-Time Scheduling: From Hard to Soft Real-Time Systems," arXiv,

2015. [Online]. Available: https://arxiv.org/pdf/1512.01978
[11] Nicolas Navet and Françoise Simonot-Lion, “Trends in Automotive Communication Systems," Richard Zurawski. Embedded Systems Handbook: Networked Embedded Systems, 2nd ed., Taylor and Francis/CRC Press, pp. 13.1-13.24, 2009, Industrial Information Technology Series, ISBN 978-1-4398-0761-3. [Online]. Available: https://inria.hal.science/inria-00439105/document
[12] L.M. Pinho et al., "Reliable real-time communication in CAN networks," in IEEE Transactions on Computers, vol. 52, no. 12, pp. 1594-1607, Dec. 2003, doi: 10.1109/TC.2003.1252855. [Online]. Available: https://ieeexplore.ieee.org/document/1252855
[13] Tommaso Cucinotta et al., "A Real-Time Service-Oriented Architecture for Industrial Automation," in IEEE Transactions on Industrial Informatics, vol. 5, no. 3, pp. 267-277, Aug. 2009, doi: 10.1109/TII.2009.2027013. [Online]. Available: https://ieeexplore.ieee.org/document/5173504
[14] PAUL POP et al., "Analysis and optimization of distributed real-time embedded systems." Proceedings of the 41st annual Design Automation Conference. 2004. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/996566.1142984