



Worst-Case Execution Time Analysis for Mixed-Criticality Tasks on Multicore Automotive Controllers: A Review of Methods and Open Problems

Julien Gumede^{1*}, Bekzat Tadesse², Hiroshi Campbell³

¹Department of Electrical Electronics, Tagus Graduate Institute of Electrical Electronics, Trinidad and Tobago

²Department of Electrical Electronics, Vistula Graduate Institute of Electrical Electronics, Hungary

³Department of Electrical Electronics, Nile Delta Center for Electrical Electronics, Thailand

Article Info

Volume 1, Issue 1, 2026

doi: [10.62715/ijesd.2026.v1.i1.pp1-7](https://doi.org/10.62715/ijesd.2026.v1.i1.pp1-7)

Received: 08 November 2025

Accepted: 01 January 2026

Published: 13 March 2026

Abstract

Automotive electronic control units now run braking, steering and powertrain functions beside diagnostics and comfort software on the same multicore device. Scheduling such a mix safely depends on a bound for how long each task can run, and that bound, the worst-case execution time, has become harder to obtain since cores began to share caches, buses and memory controllers. This narrative review brings together three bodies of work that are usually read apart: single-core timing analysis by static, measurement-based and probabilistic means; the mixed-criticality scheduling model that assigns a task several execution time budgets at different levels of assurance; and the research on interference between cores and the mechanisms that contain it. The review describes what each method assumes, where it breaks down on shared hardware, and how far it matches the freedom-from-interference requirement of ISO 26262. The literature points to a practical consensus: on current controllers a defensible bound comes from limiting interference by design and then analysing each core nearly in isolation, with measurement used to confirm the assumptions. Open problems include undocumented hardware behaviour, the gap between the academic mixed-criticality model and certification practice, and the evidential standing of probabilistic bounds.

Keywords: Worst-case execution time, Mixed-criticality systems, Multicore interference, ISO 26262, Automotive ECU, Timing analysis

© 2026 Julien Gumede et al. This is an open access article under the CC BY license (<https://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

1. Introduction

A hard real-time task is correct only if it finishes on time, and a schedulability test can say whether it will only if it is given an upper bound on the time the task needs. Since the rate-monotonic and earliest-deadline analyses of Liu and Layland (1973), that bound has been treated as an input, a single number per task called the worst-case execution time (WCET). Producing the number is a research field of its own. The survey by Wilhelm et al. (2008) remains the standard account of how it is done on a single processor core and of why the problem is hard even there.

* Corresponding author: Julien Gumede, Department of Electrical Electronics, Tagus Graduate Institute of Electrical Electronics, Trinidad and Tobago. E-mail: julien.gumede@tagus.edu

© 2026 Julien Gumede et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Two developments have complicated the picture for vehicle electronics. The first is consolidation. Functions that once had a control unit each are being gathered onto fewer and more capable devices, so a single controller may host software rated at the highest automotive safety integrity level next to software with no safety rating at all. The second is the move to multicore microcontrollers and, in domain and zone controllers, to application-class systems-on-chip. On these devices the time a task takes depends on what the other cores are doing, because cores compete for caches, buses, flash and memory controllers.

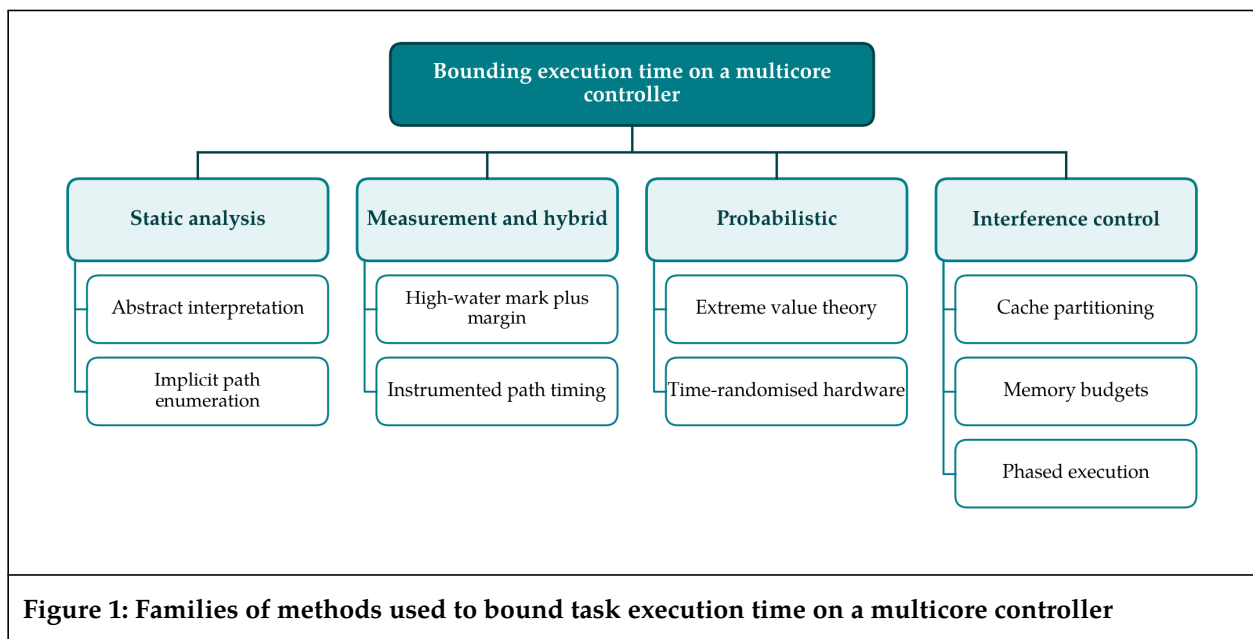
The research response has come from two directions. Mixed-criticality scheduling theory, which began with Vestal (2007), asks how tasks of different importance can share a processor when their execution time bounds are known with different degrees of confidence. Multicore timing verification asks how a bound can be obtained at all in the presence of interference from other cores (Maiza et al., 2019). This review reads the two together with the automotive setting in mind, and asks what an engineer responsible for a multicore controller can take from them.

2. Scope and Approach

This is a narrative review of the published literature. It draws on the established surveys of timing analysis, mixed-criticality scheduling and multicore verification, on the primary papers those surveys treat as foundational, and on the functional safety standard for road vehicles. Papers were chosen for their influence on the field and their relevance to automotive controllers, not through a formal search protocol, and the review makes no claim to completeness. Avionics sources are included where the automotive literature is thin, since certification authorities in that sector addressed multicore interference early and in explicit terms.

3. Execution Time Bounds on a Single Core

Three families of method estimate the WCET of a task running alone. Figure 1 places them beside the interference control techniques discussed later, and Table 1 summarises how each fares once cores share hardware.



3.1. Static analysis

Static analysis derives a bound from the executable without running it. The control flow graph is reconstructed from the binary, a value analysis determines loop bounds and the addresses of memory accesses, and a micro-architectural analysis bounds the time of each basic block given the possible states of the pipeline and caches. The mathematical basis is abstract interpretation (Cousot and Cousot, 1977), applied to caches in the must and may analyses of Ferdinand and Wilhelm (1999). The longest path is

then found by implicit path enumeration, which encodes the flow constraints as an integer linear program (Li and Malik, 1995). The result is a safe bound, provided the hardware model is right.

That proviso carries much of the weight. A sound model needs detailed knowledge of the processor, and the analysis must cope with timing anomalies, situations in which a locally faster outcome such as a cache hit leads to a longer overall execution (Lundqvist and Stenstrom, 1999). Where anomalies are possible the analysis cannot simply follow the locally worst case and has to track many hardware states, which costs both time and precision.

3.2. Measurement-based and hybrid analysis

Industrial practice leans heavily on measurement: the task is run under test, the longest observed time is recorded as a high-water mark, and an engineering margin is added. The approach needs no hardware model and works on any processor. Its weakness is that the worst path and the worst hardware state may never be exercised, so the result is an estimate and not a bound, and the size of the margin is a matter of judgement. Hybrid methods measure short program segments on the target and combine those timings using the program structure obtained statically, which removes the need to drive the longest path end to end but leaves the dependence on having seen the worst hardware state (Wilhelm et al., 2008).

3.3. Probabilistic analysis

Probabilistic timing analysis replaces the single bound with a distribution, the probabilistic WCET, which gives for each execution time budget the probability that it is exceeded. The idea goes back at least to Bernat et al. (2002). Its measurement-based form fits an extreme value distribution to observed times and extrapolates to very small exceedance probabilities. The fit is valid only if the observations satisfy the statistical assumptions of the theory, and much of the research has concerned how to make hardware behave that way, for instance through randomised cache placement, and how to check representativeness. Two comprehensive surveys set out the methods and their unresolved questions (Cazorla et al., 2019; Davis and Cucu-Grosjean, 2019).

Table 1: Approaches to bounding execution time and their standing on shared multicore hardware

Approach	Basis of the bound	Main strength	Difficulty on multicore
Static analysis	Abstract model of processor plus path search	Safe by construction if the model is sound	Joint state of all cores is intractable; models often unavailable
High-water mark	Longest observed time plus margin	Simple; needs no hardware model	Worst contention scenario is hard to provoke in test
Hybrid analysis	Measured segments combined by program structure	Avoids end-to-end worst path testing	Segment times still depend on co-runner behaviour
Probabilistic analysis	Extreme value fit to measurements	Quantified exceedance probability	Representativeness of observations is hard to argue
Isolation plus single-core analysis	Interference bounded by design, then analysed per core	Reuses mature tools; fits partitioning in standards	Residual channels and performance cost
Interference-aware response time analysis	Explicit demand on each shared resource	Avoids padding every task for full contention	Needs resource access counts per task

4. The Mixed-Criticality Model and Its Reception

Vestal observed that the more rigorously a WCET is established, the larger it tends to be. A figure acceptable for a low-criticality function may be a measured maximum, while a certification authority may insist on a far more conservative bound for a flight or braking function. His model therefore gives each

task a set of execution time budgets, one per criticality level, and asks that each task be schedulable when every task of equal or higher criticality uses the budget defined at that level (Vestal, 2007). The system runs in a low-criticality mode so long as budgets hold. If a high-criticality task overruns its optimistic budget, the system switches mode and protects the critical tasks, in the original formulations by abandoning the less critical ones.

A large literature followed. Response time analysis was extended to the model, with the adaptive mixed-criticality scheme among the best known results (Baruah et al., 2011), and multiprocessor variants were built on the earlier theory of global and partitioned scheduling (Davis and Burns, 2011). The survey by Burns and Davis (2017) catalogues this work and is the natural entry point to it.

The model has also been criticised from the standpoint of safety engineering. Ernst and Di Natale (2016) argue that it conflates criticality, a property assigned through hazard analysis, with the confidence placed in a timing estimate, and that standards require sufficient independence between functions of different integrity, which a scheme that sheds low-criticality work on demand does not obviously provide. Esper et al. (2015) reach a similar view after comparing the model's assumptions with industrial standards. The automotive standard is a case in point. ISO 26262 lets elements with different safety integrity levels coexist only if freedom from interference is shown, and it lists timing and execution among the interference classes (ISO, 2018). A task without a safety rating may be important to the driver all the same, and abandoning it at a mode change is a design decision that needs its own justification.

5. Interference on Shared Hardware

On a multicore device the execution time of a task is no longer a property of the task and its core alone. Cores may share a last-level cache, so that one task evicts another's data. They share the interconnect and the memory controller, so that accesses queue. On microcontrollers they commonly share program flash and peripheral buses. Reports from avionics, where the question was examined early, describe slowdowns of several-fold for memory-intensive code when other cores are loaded (Nowotsch and Paulitsch, 2012). The certification position paper CAST-32A asks applicants to identify every interference channel and to show that each is mitigated or bounded (CAST, 2016), and the reasoning carries over directly to road vehicles.

A fully integrated static analysis would model all cores at once. The state space makes this impractical for realistic software, and the survey literature treats it as a research direction more than a usable technique (Maiza et al., 2019). The alternatives fall into two groups: control the interference so that single-core analysis remains valid, or account for it explicitly in the schedulability analysis.

5.1. Controlling interference

Shared caches can be partitioned. Page colouring in the operating system assigns each core, or each task, a disjoint set of cache sets, and the frameworks of Mancuso et al. (2013) and Kim et al. (2013) show how this can be done on commodity platforms without hardware changes. Memory bandwidth can be budgeted: MemGuard uses performance counters to limit the number of memory accesses a core may issue per regulation period and suspends the core when its budget is spent (Yun et al., 2013). The predictable execution model goes further by restructuring tasks into a memory phase, in which code and data are loaded into private cache, and an execution phase that touches no shared memory; a system-wide schedule then lets only one core be in its memory phase at a time (Pellizzoni et al., 2011). Each technique trades average performance for a contention bound that can be stated in advance.

5.2. Accounting for interference

Where isolation is incomplete, the residual interference must enter the analysis. The simplest method pads each WCET for the worst contention on every access, which is safe and often so pessimistic that the second core brings little benefit. The multicore response time analysis framework of Altmeyer et al. (2015) avoids this by keeping resource demand separate from execution time: each task is described by how many bus, cache and memory accesses it can make, each resource by how its arbiter serves competing

requests, and the response time calculation combines the two over the interval of interest. The approach depends on being able to bound those access counts, which brings the problem back to static analysis or to carefully instrumented measurement.

6. The Automotive Setting

Classic automotive microcontrollers differ from the commodity platforms on which most interference research is done. They have short pipelines, small or no shared caches, execution from flash and local scratchpad memory, and often lockstep core pairs for fault detection. Static analysis is tractable on such cores, and interference arises mainly through shared flash, buses and peripherals. The operating system layer defined by AUTOSAR already provides partitioned fixed-priority scheduling, memory protection and timing protection in the form of execution budgets and inter-arrival monitoring, which is close to the monitoring a mixed-criticality scheme requires. A budget overrun is ordinarily treated as a fault to be contained, though, and not as a trigger for a planned change of mode.

High-performance domain controllers are another matter. They use application processors with deep cache hierarchies and out-of-order execution, usually under a hypervisor that hosts safety-rated and unrated guests. Here the commodity multicore problem appears in full, and the mechanisms reviewed above, cache partitioning and memory bandwidth regulation at the hypervisor level in particular, are the available tools. What the automotive literature lacks, compared with avionics, is an agreed statement of what timing evidence for such a platform should contain.

7. Measurement as Supporting Evidence

Whatever analysis is chosen, the assumptions behind it have to be tested on the real device. The technique that has become customary is to run the task under analysis against synthetic co-runners written to load one shared resource as heavily as possible, an approach developed for commodity multicore processors by [Radojkovic et al. \(2012\)](#). If the execution time observed under such stress stays within the bound claimed, the isolation mechanisms are doing their work; if it does not, an interference channel has been missed. Such tests cannot establish a bound on their own, because no stress program is guaranteed to produce the worst alignment of requests. Their value is as a check on the argument. Trace interfaces on automotive microcontrollers make the measurements unintrusive, and the same instrumentation serves the budget monitoring that the operating system performs in the field.

8. Open Problems

Several gaps recur across the sources. Hardware documentation is the first: sound analysis requires knowledge of arbitration policies, prefetchers and controller behaviour that vendors seldom publish, and measurement can reveal such behaviour but cannot prove its absence. The second is the standing of probabilistic bounds. A probabilistic WCET sits naturally beside the failure rate targets used for hardware, yet the conditions under which an extreme value fit can be trusted remain debated ([Davis and Cucu-Grosjean, 2019](#)). The third is the distance between theory and practice in mixed-criticality scheduling: models that degrade low-criticality service gracefully, instead of abandoning it, are closer to what standards allow, and their analysis is less mature ([Burns and Davis, 2017](#)). A fourth concerns composability. Integrators need timing budgets that hold for a software component whatever is placed on the other cores, and current methods mostly deliver bounds tied to one configuration.

9. Conclusion

The literature does not offer a single method that yields tight, certifiable execution time bounds for arbitrary software on arbitrary multicore hardware. It does support a workable strategy for automotive controllers: restrict the interference channels through partitioning and budgeting, analyse each core with the static or hybrid tools that are already mature, account explicitly for whatever sharing remains, and use measurement under stress to test the assumptions. The mixed-criticality model contributes the useful idea of budgets held at different levels of assurance, though its mode-change semantics must be reconciled

with the freedom-from-interference requirement before it can be applied as published. Progress now depends as much on hardware transparency and on agreement about acceptable evidence as on new analysis techniques.

Conflict of interest

To be declared by the author(s) before publication.

Funding

To be declared by the author(s) before publication.

References

- Altmeyer, S., Davis, R.I., Indrusiak, L., Maiza, C., Nelis, V. and Reineke, J. (2015). [A generic and compositional framework for multicore response time analysis](#). *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS)*, 129-138.
- Baruah, S.K., Burns, A. and Davis, R.I. (2011). [Response-time analysis for mixed criticality systems](#). *Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS)*, 34-43.
<https://doi.org/10.1109/RTSS.2011.12>
- Bernat, G., Colin, A. and Petters, S.M. (2002). [WCET analysis of probabilistic hard real-time systems](#). *Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS)*, 279-288.
- Burns, A. and Davis, R.I. (2017). [A survey of research into mixed criticality systems](#). *ACM Computing Surveys*, 50(6): 1-37. <https://doi.org/10.1145/3131347>
- CAST (2016). [Position Paper CAST-32A: Multi-core Processors](#). Certification Authorities Software Team.
- Cazorla, F.J., Kosmidis, L., Mezzetti, E., Hernandez, C., Abella, J. and Vardanega, T. (2019). [Probabilistic worst-case timing analysis: Taxonomy and comprehensive survey](#). *ACM Computing Surveys*, 52(1): 1-35. <https://doi.org/10.1145/3301283>
- Cousot, P. and Cousot, R. (1977). [Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints](#). *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*, 238-252.
<https://doi.org/10.1145/512950.512973>
- Davis, R.I. and Burns, A. (2011). [A survey of hard real-time scheduling for multiprocessor systems](#). *ACM Computing Surveys*, 43(4): 1-44. <https://doi.org/10.1145/1978802.1978814>
- Davis, R.I. and Cucu-Grosjean, L. (2019). [A survey of probabilistic timing analysis techniques for real-time systems](#). *Leibniz Transactions on Embedded Systems*, 6(1): 03:1-03:60.
- Ernst, R. and Di Natale, M. (2016). [Mixed criticality systems: A history of misconceptions?](#) *IEEE Design & Test*, 33(5): 65-74. <https://doi.org/10.1109/MDAT.2016.2594790>
- Esper, A., Nelissen, G., Nelis, V. and Tovar, E. (2015). [How realistic is the mixed-criticality real-time system model?](#) *Proceedings of the 23rd International Conference on Real Time and Networks Systems (RTNS)*, 139-148.
- Ferdinand, C. and Wilhelm, R. (1999). [Efficient and precise cache behavior prediction for real-time systems](#). *Real-Time Systems*, 17(2-3): 131-181. <https://doi.org/10.1023/A:1008186323068>
- ISO (2018). [ISO 26262: Road Vehicles, Functional Safety, 2nd edition](#). International Organization for Standardization, Geneva.
- Kim, H., Kandhalu, A. and Rajkumar, R. (2013). [A coordinated approach for practical OS-level cache management in multi-core real-time systems](#). *Proceedings of the 25th Euromicro Conference on Real-Time Systems (ECRTS)*, 80-89.
- Li, Y.-T.S. and Malik, S. (1995). [Performance analysis of embedded software using implicit path enumeration](#). *Proceedings of the 32nd ACM/IEEE Design Automation Conference (DAC)*, 456-461.

- Liu, C.L. and Layland, J.W. (1973). [Scheduling algorithms for multiprogramming in a hard-real-time environment](#). *Journal of the ACM*, 20(1): 46-61. <https://doi.org/10.1145/321738.321743>
- Lundqvist, T. and Stenstrom, P. (1999). [Timing anomalies in dynamically scheduled microprocessors](#). *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS)*, 12-21.
- Maiza, C., Rihani, H., Rivas, J.M., Goossens, J., Altmeyer, S. and Davis, R.I. (2019). [A survey of timing verification techniques for multi-core real-time systems](#). *ACM Computing Surveys*, 52(3): 1-38. <https://doi.org/10.1145/3323212>
- Mancuso, R., Dudko, R., Betti, E., Cesati, M., Caccamo, M. and Pellizzoni, R. (2013). [Real-time cache management framework for multi-core architectures](#). *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 45-54.
- Nowotsch, J. and Paulitsch, M. (2012). [Leveraging multi-core computing architectures in avionics](#). *Proceedings of the Ninth European Dependable Computing Conference (EDCC)*, 132-143.
- Pellizzoni, R., Betti, E., Bak, S., Yao, G., Criswell, J., Caccamo, M. and Kegley, R. (2011). [A predictable execution model for COTS-based embedded systems](#). *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 269-279.
- Radojkovic, P., Girbal, S., Grasset, A., Quinones, E., Yehia, S. and Cazorla, F.J. (2012). [On the evaluation of the impact of shared resources in multithreaded COTS processors in time-critical environments](#). *ACM Transactions on Architecture and Code Optimization*, 8(4): 1-25.
- Vestal, S. (2007). [Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance](#). *Proceedings of the 28th IEEE International Real-Time Systems Symposium (RTSS)*, 239-243. <https://doi.org/10.1109/RTSS.2007.47>
- Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J. and Stenstrom, P. (2008). [The worst-case execution-time problem: Overview of methods and survey of tools](#). *ACM Transactions on Embedded Computing Systems*, 7(3): 1-53. <https://doi.org/10.1145/1347375.1347389>
- Yun, H., Yao, G., Pellizzoni, R., Caccamo, M. and Sha, L. (2013). [MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms](#). *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 55-64. <https://doi.org/10.1109/RTAS.2013.6531079>