Electronic Design

By MARK PITCHFORD, Technical Specialist, LDRA,
https://ldra.com

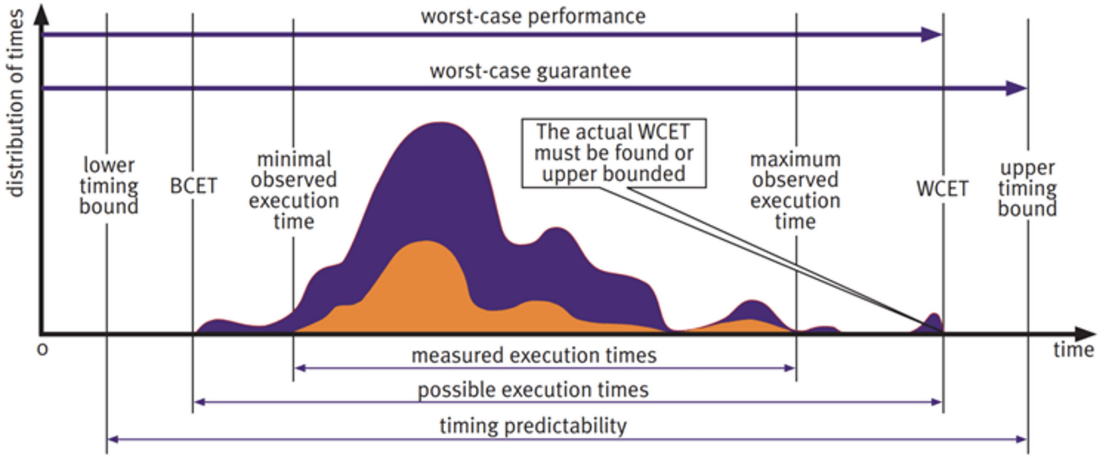
A Holistic Approach to Meet Multicore-System Timing and Interference Requirements

This article offers insights on how different analysis techniques combined with CAST-32A and A(M)C 20-193 guidance can ensure confidence in meeting timing requirements in real-time multicore systems.

Multicore processors (MCPs) address limitations in single-core computing by overcoming the physical limitations of clock speeds and temperature control to support larger central processing unit (CPU) workloads. For hard real-time applications, such systems pose challenges to developers in knowing how to meet strict timing requirements and managing interference among heterogenous cores.

Hard real-time applications demand deterministic execution times. While the average execution time for a given set of tasks running on an MCP platform tends to be lower than for the same set of tasks on a single-core processor (SCP) setup, the distribution of those times is often spread out. This variability makes it difficult to guarantee timing requirements for critical tasks. It also causes significant issues for applications where every individual execution time matters—not just the average.

The Certification Authorities Software Team (CAST) addresses these challenges in its position paper for avionics

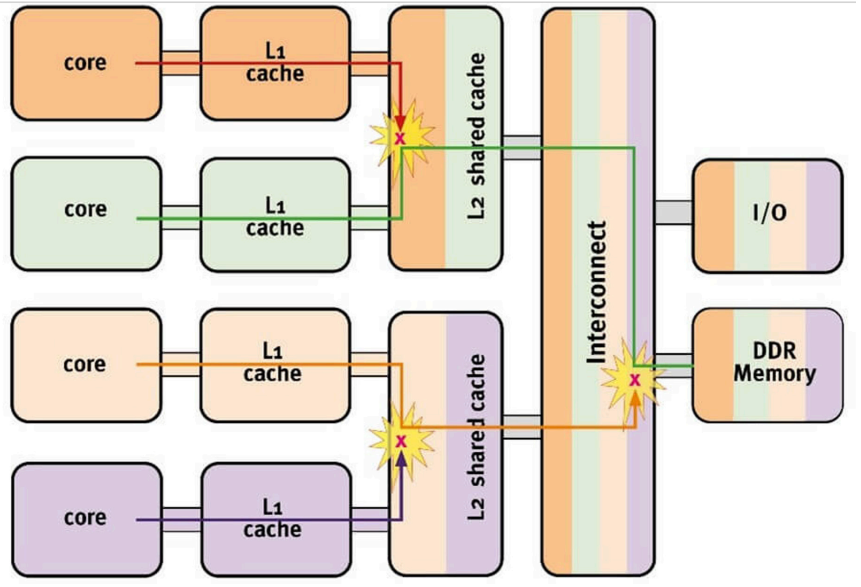


WCET: Worst-Case Execution Time
BCET: Best-Case Execution Time
ACET: Average-Case Execution Time

1. Different execution times and guarantees for a given real-time task.

manufacturers, "CAST-32A, Multicore Processors." This paper sets out a series of objectives that must be met in the software development lifecycle (SDLC) to ensure that a multicore system is understood, particularly timing behaviors. These objectives aren't prescriptive requirements for developers to implement, but rather guide and support activities toward generally accepted standards like DO-178C.

The AMC 20-193 document has already superseded and deprecated CAST-32A in Europe, and it's expected that AC 20-193 will soon follow suit in the U.S. Known collectively as A(M)C 20-193, these successor documents very largely duplicate CAST-32A.



2. Here's one example of hardware interference in a shared hierarchical memory architecture.

This article describes approaches for applying the guidance provided in CAST-32A and its successor documents, including techniques for measuring timing and interference on heterogeneous multicore processors.

Why Worst-Case Execution Times Matter for Hard Real-Time Systems

Hard real-time applications, in avionics or elsewhere, must meet strict timing requirements to ensure operational functionality and safety. Soft real-time systems are less constrained and the impacts of missing a timing deadline are less severe. The *table* provides a real-world example of the key differences between these two types of real-time systems.

The shortest execution time for a given CPU task is called the best-case execution time (BCET); the longest is called the worst-case execution time (WCET). *Figure 1* provides a visual of how these values are determined given an example set of timing measurements.

SCPs can guarantee the required upper timing bounds are always met as long as extra CPU capacity is planned and maintained sufficiently. MCPs present a greater challenge because there's no effective method for calculating a guaranteed tasking schedule that accounts for multiple processes running in parallel across multiple heterogenous cores. Compounding this challenge is the impact of hardware interference on tasks, which can change from core to core to disrupt the predictability and measurement of task timing.

For example, in the shared hierarchical memory architecture *(Fig. 2)*, memory-access collisions cause different interference channels for different cores. These channels may result in the spreading of distribution of task execution times, making it difficult to guarantee predictable and hard real-time determinism in the system.

Unlike single-core systems, developers have no multicore approximation methods that can be calculated statically to generate usable approximations of WCETs and BCETs. Developers must leverage test and measurement results that offer as much confidence as possible.

HARD VS. SOFT REAL-TIME SYSTEMS

	Autopilot system (hard real-time)	In-flight entertainment (soft real-time)
Timing requirements	Deterministic—must demonstrate predictability and adherence to strict timing constraints	Probabilistic—less restrictive, may be "best effort"
Impact of missing a timing deadline	Can compromise mission and/or passenger safety	May diminish user experience but doesn't compromise passenger safety
System effectiveness if a timing deadline is missed	Immediate reduction in performance and/or operational capabilities	Gradual reduction in performance and/or operational capabilities

How CAST-32A and A(M)C 20-193 Guide Developers

CAST-32A and A(M)C 20-193 supplement the DO-178C processes for multicore processors with 10 objectives

Halsteads (*Cashregister.c*)

File	Total Operators	Total Operands	Unique Operators	Unique Operands	Vocabulary	Length	Volume
Total for Cashregister.c	149	230	16	56	72	379	2338

3. Halstead's metrics are calculated using the LDRA tool suite. (Source: LDRA)

for software planning through to verification. These objectives include the following (specific objectives are identified in parentheses):

- MCP configuration settings are determined and documented, as these options can impact system behavior and safety in many ways. It's especially important to track these settings throughout the project lifecycle, because the nature of iterative development and testing makes it likely that configurations will change. (MCP_Resource_Usage_1)
- MCP interference channels are identified and mitigation strategies are verified to reduce the likelihood of issues at runtime. (MCP_Resource_Usage_3)
- MCP tasks have sufficient time to complete execution and adequate resources are allocated when hosted in the final deployed configuration. (MCP_Software_1 and MCP_Resource_Usage_4)
- Data and control coupling between software components have been exercised during testing to demonstrate that their impacts are restricted to those intended by the design. This is important because flawed coupling is likely to have an impact on timing, as data is accessed at different times by different threads on different cores. (MCP_Software_2)

In terms of resourcing, CAST-32A and A(M)C 20-193 recognize the value of partitioning in time and space. This enables developers to perform application verification and WCET determination separately if they have verified that

4. LDRA tool suite screenshots show histograms of execution times and timing summaries. (Source: LDRA)

the MCP platform provides robust resource and time partitioning. As the documents imply, the use of robust partitioning is likely to simplify interference mitigation.

Neither CAST-32A nor A(M)C 20-193 prescribe methods for achieving these objectives, leaving them to the development team to select. While these documents are advisory guidelines only, developers should be aware that section 6.3.4 of the DO-178C standard explicitly specifies the need for WCET analysis:

6.3.4f: Accuracy and consistency: The objective is to determine the correctness and consistency of the Source Code, including stack usage, memory usage, […] worst-case execution timing, exception handling […]

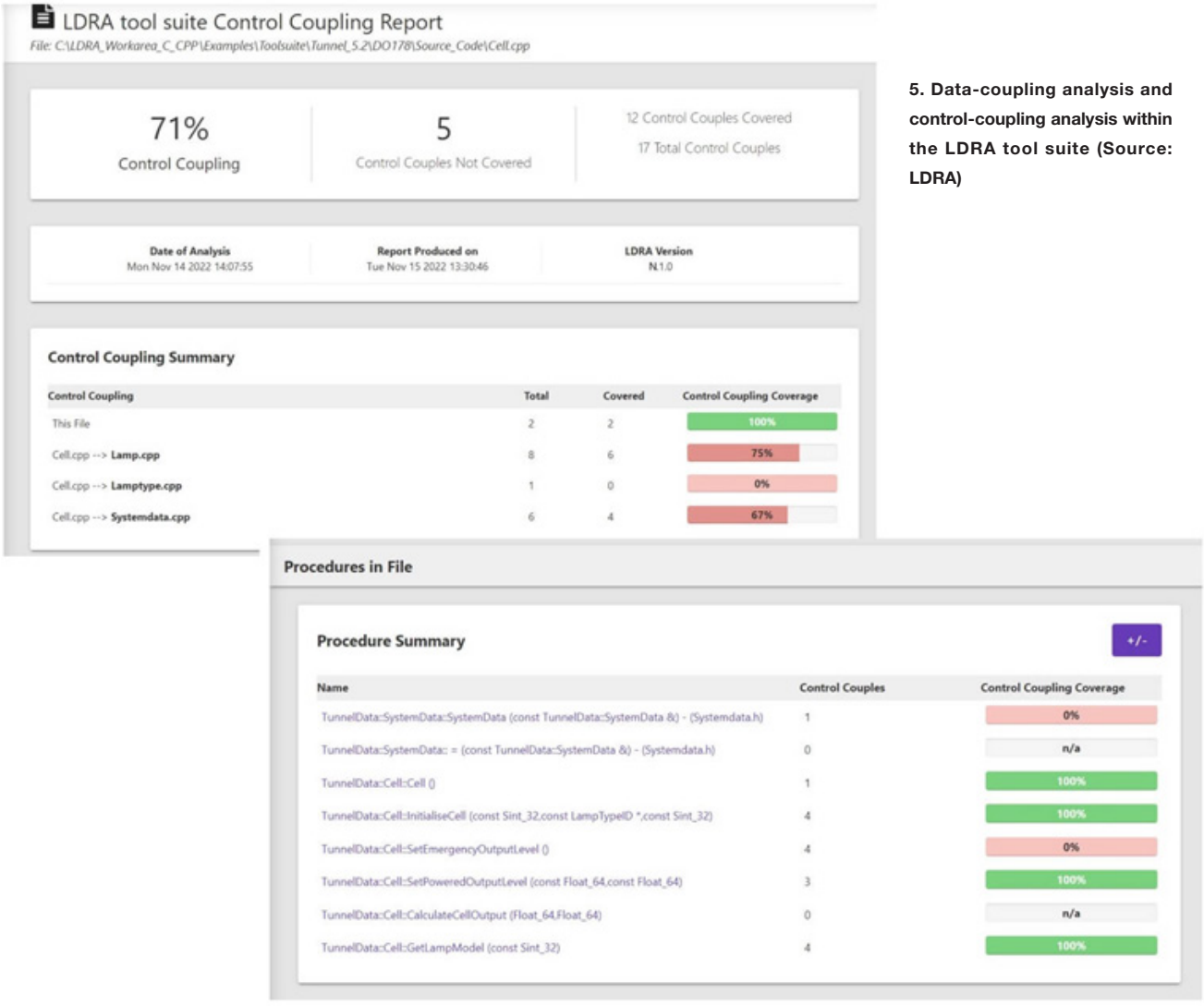
Analyzing Execution Times for Real-Time Systems

The state space variability of MCP-based systems has implications for timing measurements across the development lifecycle. The four execution-time analysis methods that follow are proven effective to meet the needs of deterministic task schedule design and support the CAST-32A and A(M)C 20-193 guidelines.

Prioritizing timing analysis using static analysis

The Halstead complexity metrics can be used to identify sections of code that pose the highest demands on processing time, and static-analysis tools are able to both calculate and supplement this data with actual measurements from the target system. Such metrics provide an early indication of the complexity and potential resource requirements of a code segment by providing insights into module size, control-flow structures, and data flow.

Using Halstead's metrics helps developers prioritize their efforts in timing analysis and in optimizing sections of code



5. Data-coupling analysis and control-coupling analysis within the LDRA tool suite (Source: LDRA)

that exhibit larger size, higher complexity, and intricate data-flow patterns. This helps reduce the risk of timing violations and improve the overall timing behavior of the system. Tools like the LDRA tool suite can calculate these metrics *(Fig. 3)*.

Empirical execution-time analysis

Once developers identify the high-priority modules, they can use dynamic analysis to measure and report on task timings as well as tune the system if objectives aren't met ("interference analysis"). To ensure the utmost accuracy, they should be aware of three critical considerations:

1. The analysis must take place in the environment where the application will eventually run to avoid the influence of configuration differences between development and production, such as compiler options, linker options, and hardware features.

2. The analysis must execute enough tests repeatedly to account for environmental and application variations between runs.

3. Automation is highly recommended to ensure a sufficient number of tests are executed in a reasonable amount of time, as well as avoid the influence of relatively slower manual actions.

The LDRA tool suite offers a particularly robust mechanism for execution time analysis via a "wrapper" test harness to exercise modules on the target device *(Fig. 4)*. This mechanism automates timing measurements and allows developers to define the components under test, whether at the function level, a subsystem of components, or the overall system. It also enables developers to specify the type of CPU stress tests to perform to improve confidence in the results, such as using the open-source Stress-ng workload generator.

Control and data-coupling analyses

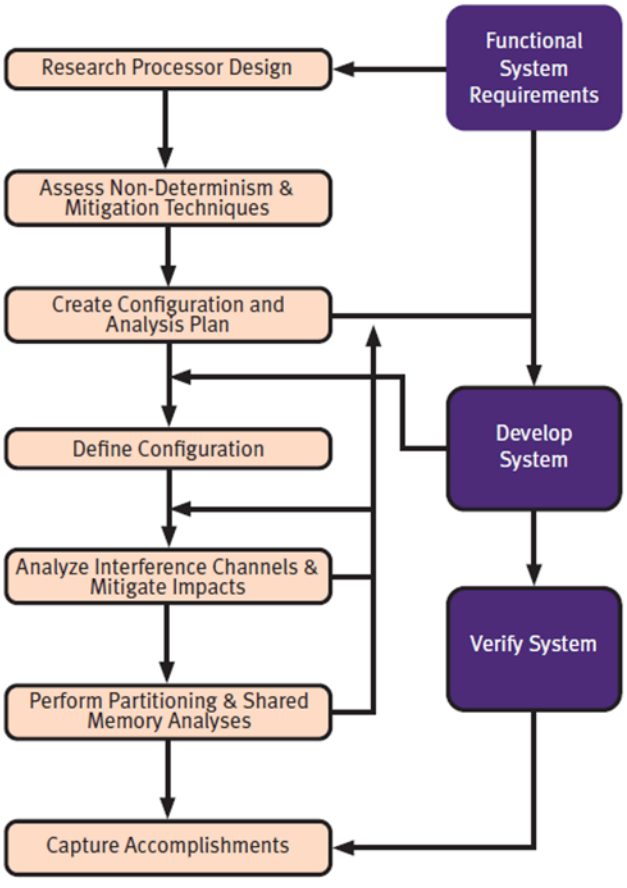
Control and data-coupling analyses examine how task execution and data dependencies in one task affect another task to identify potential timing issues. For example, if one task relies on the completion of another task before it can proceed, there may be issues when execution is delayed. If the currently executing task misses its deadline, it can impact the timing requirements of the dependent task.

Similarly, data-coupling issues may arise when multiple tasks access a shared resource concurrently without proper synchronization mechanisms. This will ultimately lead to data races or resource contention issues that impact timing.

The LDRA tool suite supports control coupling and data-coupling analyses relating to data and control flow between software components and applications *(Fig. 5)*. The results of these analyses help developers to identify critical sections of code that require optimization or restructuring to minimize control-flow dependencies and alleviate data contention.

Requirements traceability

The empirical nature of timing analysis in MCP environments makes interference analysis an iterative process *(Fig. 6)*. The repeated modification of tasks and tuning of configuration parameters will likely have knock-on effects to other aspects of test that will need to be regressed. Conversely, any requirements change may introduce or alter interference channels in the system. Therefore, knowing where and how impacts may occur helps developers to avoid downstream issues.



6. This is the recommended approach to multicore timing-analysis feedback. (Source: LDRA, based upon work by Wind River Systems and Collins Aerospace)

It's also important for developers to track the fulfillment of CAST-32A, A(M)C 20-193, DO-178C, and other applicable guidance and to regress tests that show their continued fulfillment.

An automated mechanism, such as the requirements traceability features of the LDRA tool suite, helps developers keep track of the components that need revisiting based on the results of timing verification and validation.

Conclusion

Ensuring timing requirements are met in hard real-time multicore systems is a complex undertaking. The influences

on task timing and analysis are numerous and require a disciplined approach to test setup and execution. In addition, the tight coupling between task execution time and software requirements means development teams must employ a robust feedback loop to ensure test failures trigger the appropriate corrective actions.

By addressing such challenges, developers are better able to leverage the capabilities of MCPs in robust, reliable, and certifiable ways.

Mark Pitchford has over 25 years' experience in software development for engineering applications. He has worked on many significant industrial and commercial projects in development and management, both in the UK and internationally. Since 2001, he has worked with development teams looking to achieve compliant software development in safety and security critical environments, working with standards such as DO-178, IEC 61508, ISO 26262, IIRA and RAMI 4.0.

Mark earned his Bachelor of Science degree at Trent University, Nottingham, and he has been a Chartered Engineer for over 20 years. He now works as Technical Specialist with LDRA Software Technology.