LDRA
Software Technology

Technical White Paper

Developing compliant critical software systems with multicore processors

www.ldra.com

Contents

Introduction

Most of us are familiar with multicore processors and the benefits they have brought to our daily lives. They have been available in personal computers since the early 2000s [1], and NVIDIA was promoting their benefits in mobile devices as long ago as 2010 [2]. Multicore designs address the problem of processors hitting the ceiling of their physical limitations in terms of their clock speeds and how effectively they could be cooled and still maintain accuracy. By moving to extra cores on a single processor chip, manufacturers avoided problems with the clock speeds by effectively multiplying the amount of data that could be handled by the Central Processing Unit (CPU).

With these principles established, it wasn't long before the early two core designs were supplemented by options for four, six and even ten or more cores. In the early designs, all cores were always identical to each other – that is, they were homogenous (or symmetrical) MultiCore Processors (MCPs).

These processors are built for SMP (Symmetric MultiProcessing) operating systems. These operating systems schedule processes across the cores in order to balance the load – an approach used by Windows, Linux, iOS, and Android to leverage the capabilities of MCPs.

A task may be defined as a contiguous sequence of code within a thread of execution. Computing systems ranging from laptop PCs and mobile devices right through to industrial control systems, automotive systems and aeronautical applications are designed to run one or more tasks concurrently.

The key distinguishing factor here lies in the response times to stimuli that are acceptable in their different operational environments.

Laptops and mobile devices deploy standard control processing, where the control process runs at a particular speed with no deadline. Increasing the performance of these systems is a simple matter of increasing the speed of the processor.

Real-time control systems are closed loop, allowing only a tight time window to gather data, process that data, and update the system. Hardware and software architectures have also evolved, potentially impacting the calculation and management of those time windows. Whatever the architecture, if the time window is missed then the stability of the system is degraded – and that can be catastrophic to critical applications.

The industrial pyramid

In practice, the criticality of this timing window is not a simple dichotomy between closed- and open-loop control systems. For example, the industrial pyramid in Figure 1 represents a hierarchical control architecture typical of manufacturing environments.

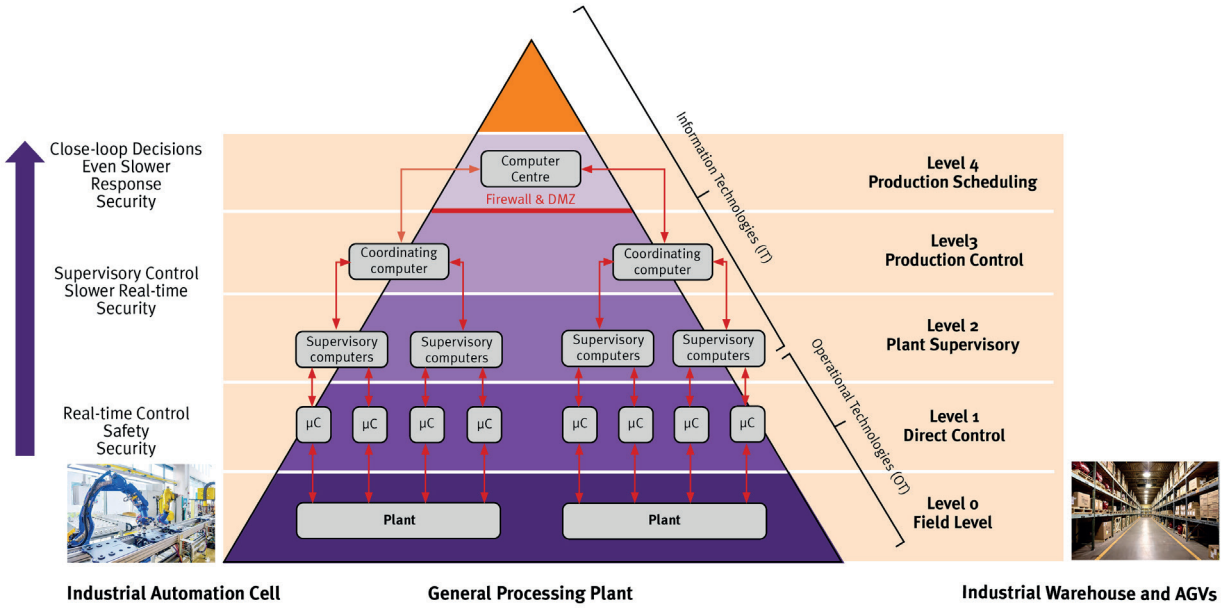


Figure 1: The industrial (or automation) pyramid

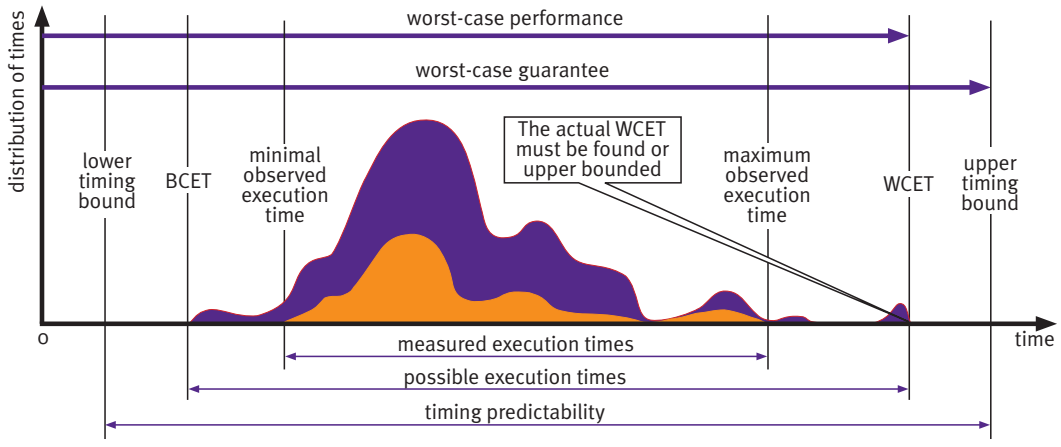
It classifies the different Information Technology (IT) and Operational Technology (OT) layers (known as "levels") of industrial automated production plants. Every level has its own tasks and infrastructure. One representation of the industrial pyramid is shown in Figure 1. Level 0 (field level) is closest to the devices and sensors, whereas Level 4 (production scheduling) is the furthest from the manufacturing floor.

Characteristics of devices and systems vary accordingly. For systems at level 0, safety, security, and tight real-time control loops (uSecs to mSecs) are paramount. The supervisory control systems at level 2 require slower but still deterministic loops (multiple msecs to secs). Level 3 and 4 systems have less demanding real-time requirements where open-loop systems are likely to be adequate.

The criticality of worst-case execution times

As illustrated by this example, hard real-time systems need to satisfy stringent timing constraints which are derived from the systems they control. In general, upper bounds on the execution times are needed to show the satisfaction of these constraints. Ultimately the question is – do we have enough time to finish what we need to do before the system needs to do something else? Providing and verifying bounds allows us to verify this statistically.

Suppose that a real-time system runs on a single-core processor and consists of several tasks, running concurrently. Figure 2 depicts several relevant properties of a real-time task [3].



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

Figure 2: The WCET & BCET are the longest & shortest execution time possible for a program, respectively.

A task typically shows a certain variation of execution times depending on the input data or different behaviour of the environment. The set of all execution times is shown as the upper curve. The shortest execution time is called the Best-Case Execution Time (BCET), and the longest time is called the Worst-Case Execution Time (WCET). In most cases the state space is too large to exhaustively explore all possible executions and thereby determine the exact WCET and BCET, but there are approximations that allow these values to be estimated to a useful extent.

WCET and multicore processors

Single-core processors cannot run multiple processes in parallel, and instead use rapid scheduling to make it appear as though they do. As proved by Liu and Layland as long ago as 1973 [4], there is a very sound basis for taking such an approach. For a single core processor, multitasking real-time systems can be guaranteed to hit their deadlines as long as sufficient CPU headroom (capacity) is allowed for.

Multicore Processors (MCPs) introduce an extra level of complication in that they genuinely do run multiple processes in parallel. Unlike single processor applications, the task of finding a schedule of X tasks on Y processor cores such that all tasks meet their deadline has no efficient algorithm.

The operating systems used in laptops and mobile phones do a good job, but they cannot guarantee to meet task deadlines.

Exacerbating that problem, in multicore processors hardware interference can occur anywhere hardware is shared between processes (Figure 3). For example, often an entire hierarchical memory is shared so that interference is possible in many places. These interference channels cause the execution-time distribution to spread. Instead of a tight peak, the distribution of execution times becomes wide with a long tail [5].

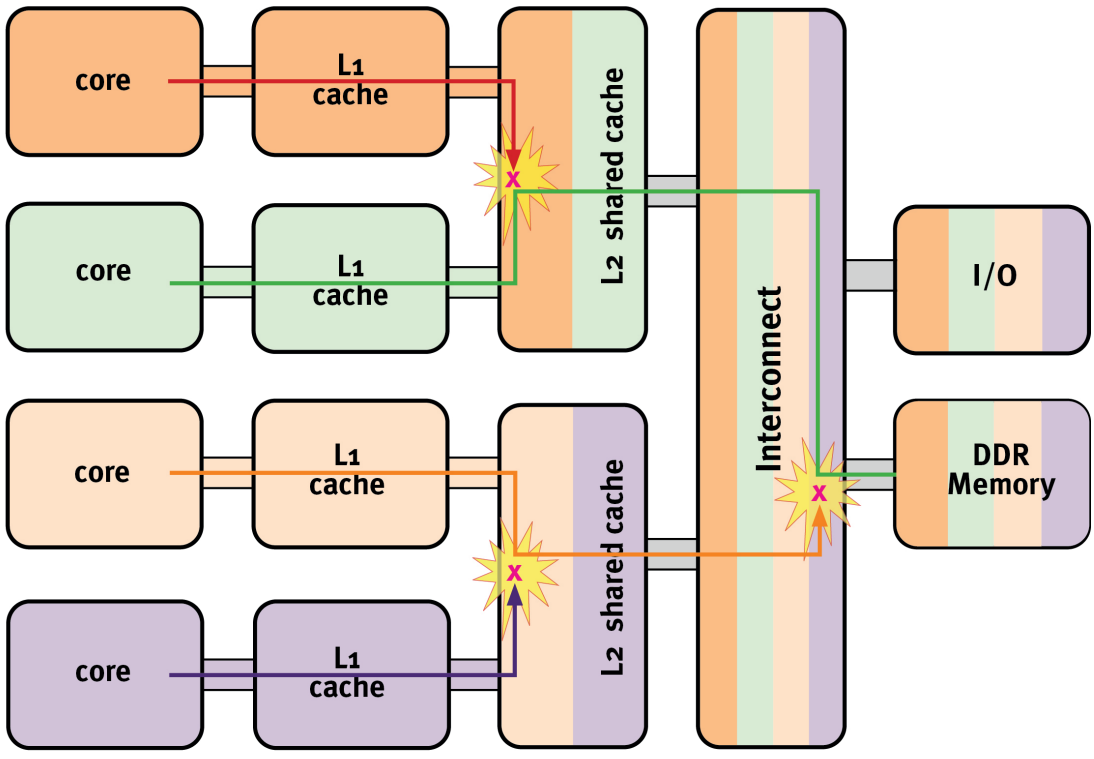


Figure 3: A representation of hardware interference [5]

Outside the realm of safety critical applications, these issues are of little consequence. But where functional safety is paramount, they are critical.

Functional safety standards and WCET

The laissez-faire approach that is entirely reasonable for a laptop or a mobile phone cannot apply here, and the standards that ensure safety across the sectors require that the issue is dealt with adequately.

Civil aviation: DO-178C, CAST-32A, AMC 20-193 and AC 20-193

For many years, the use of multicore processors in civil aviation has been restricted to the use of a single core only due to concerns over some characteristics of multicore processors. CAST-32A [6] was an advisory position paper written to address that. It was supplemental to DO-178C [7], which is the primary document by which the certification authorities approve all commercial software-based aerospace systems.

Strictly speaking, the RTCA "DOcument" series – DO-178, DO-330, DO-278 and so on – are collections of guidelines, not standards. However, they are known colloquially as standards and referenced as that here for expediency.

Similarly, although not generally classified as a functional safety standard, DO-178C does address functional safety issues.

DO-178C established a need for the analysis of WCET, highlighting it in §6.3 (Software Reviews and Analyses), §6.3.4 (Reviews and Analyses of Source Code), and §11.20 (Software Accomplishment Summary).

CAST-32A and its successor documents, AMC 20-193 [8] and AC 20-193, build on that to include detailed guidance on the criteria that are to be satisfied if multicore processors are to be deployed in DO-178C compliant applications.

With reference to interference channels, they specifically highlight that "*It is … important to identify the interference channels that could cause interference between the software applications hosted by their MCP platform, to mitigate the effects of each of those interference channels and to verify the selected means of mitigation.*"

Safety-related systems: IEC 61508

IEC 61508 [9] "Functional safety of electrical/electronic/programmable electronic safety-related systems" is widely accepted as a reference standard. Although it is often applied directly in the development of safety critical systems, its generic nature also makes it an ideal "blank canvas" for the derivation of industry and sector specific standards.

IEC 61508 part 3 §7.9 is concerned with software verification. §7.9.2.14 discusses the verification of timing performance more specifically, requiring that the "*verification of timing performance: predictability of behaviour in the time domain shall be verified." It goes on to note that "timing behaviour may include… worst case execution time*".

Automotive applications: ISO 26262

A derivative of the generic IEC 61508 standard, ISO 26262 [10] requires that timing constraints should be included within the software safety requirements. Both the worst-case execution time at the code level and the response time at the system level are to be considered, and there is reference to "*appropriate scheduling properties*".

Temporal constraints also are a part of the software architectural design (Part 6 §7.4.5), especially the worst-case execution time.

Railway and GTS systems: The EN 5012x series

Like other derivatives of the IEC 61508 standard, the EN 5012x [11] series of standards details concerns with response timing and memory constraints. EN 50128 §D.45 requires that "*An analysis is performed which will identify the distribution demands under average and worst-case conditions*".

Medical devices: IEC 62304

IEC 62304 [12] is another IEC 61508 derivative, and it is also heavily influenced by US FDA regulations [13]. IEC 62304 §5.2.2 suggests the definition of functionality and capability requirements should include "*timing requirements*".

Implications and best practices

The analysis of WCET

Even in the case of single core processors, the calculation of WCET based on first principles is not a trivial exercise. Several methods exist, including end-to-end measurements of execution times, and manual static analysis techniques such as counting and summing assembler instructions for each function, loop etc.

Although static analysis tools do exist for the purpose, the calculation of a definitive value of WCET by mathematical analysis is not soluble in the general case. According to Reinhard Wilhelm et al., any such approach will therefore require approximations to be applied which have to be correct, but not necessarily complete [3]. The result is that such tools will necessarily err "on the safe side" which is better than nothing, but in an environment where precision is everything, it cannot be ideal – even without the additional vagaries introduced by MCPs.

However, there are long standing and proven mechanisms available to measure the properties of software code which are independent of their execution on any particular platform. For example, Halstead's metrics [14] reflect the implementation or expression of algorithms in different languages to evaluate the software module size, software complexity, and the data flow information – and these can be calculated precisely from the static analysis of the source code (Figure 4). Such an approach can identify which sections of code are the most demanding of processing time but cannot provide absolute values for maximum time elapsed.

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

Figure 4: Halstead's Metrics as calculated using the TBvision component of the LDRA tool suite [15]

The same static analysis also yields call diagrams, presenting a means of visualizing where the most demanding functions revealed by this analysis are exercised in the context of the whole code base (Figure 5).

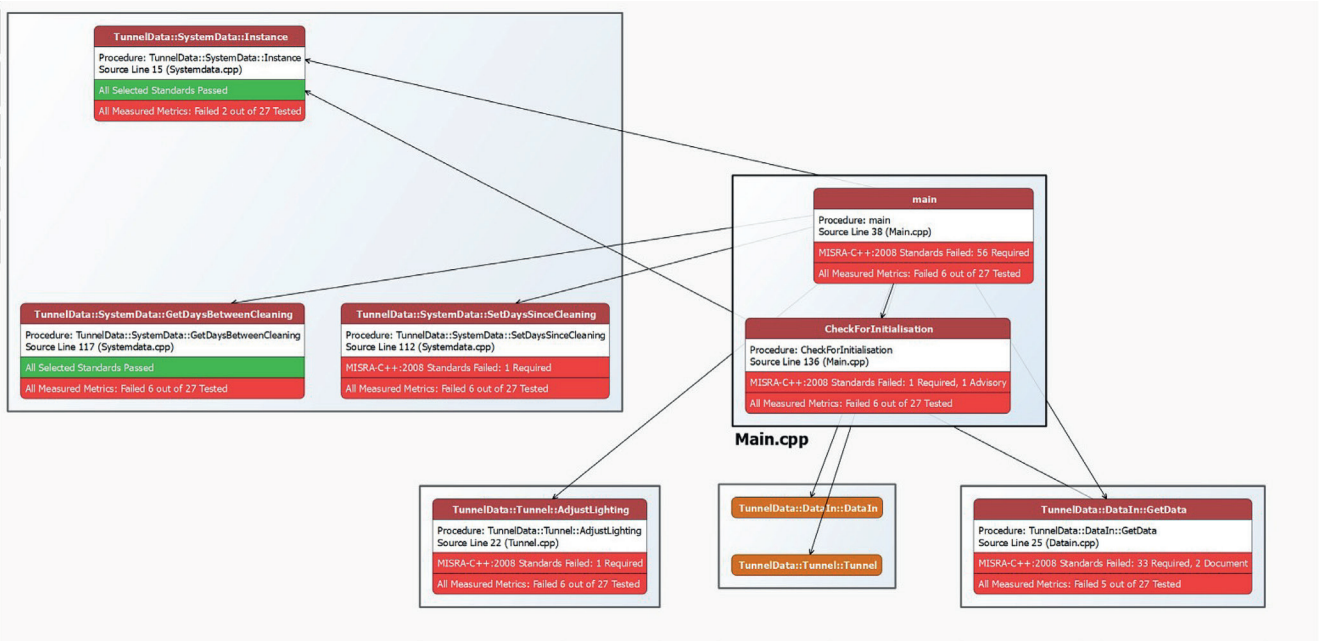


Figure 5: Call diagrams generated by the TBvision component of the LDRA tool suite provide a means to visualize where the most demanding functions are called

There is no better way to ascertain how that information translates into time elapsed for a particular function or call tree than by measuring it in the environment in which it is intended to run. That can be measured dynamically by deploying the TBrun component of the LDRA tool suite, complete with the supplementary TBwcet module (Figure 6).

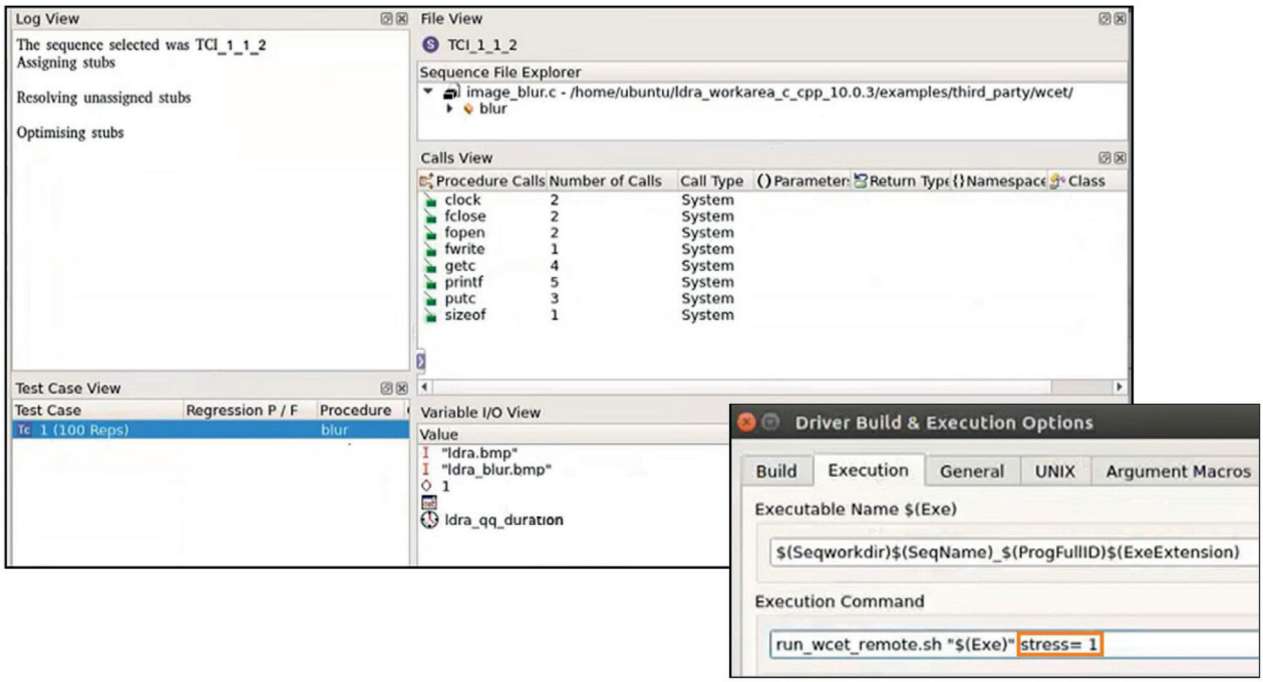
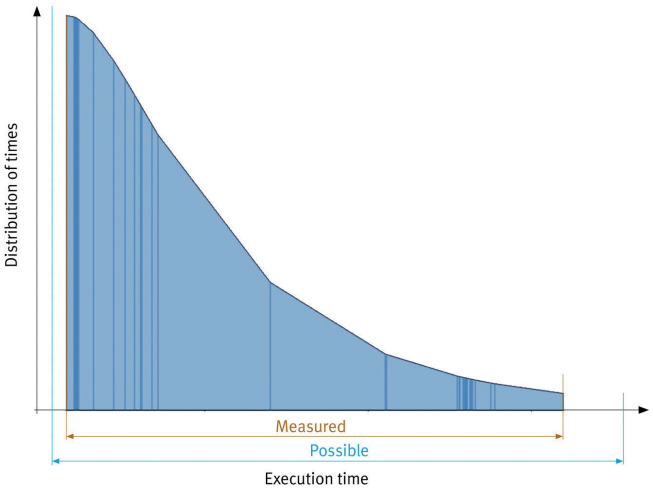


Figure 6: Using the TBrun [16] component of the LDRA tool suite with the TBwcet [17] module to measure execution times

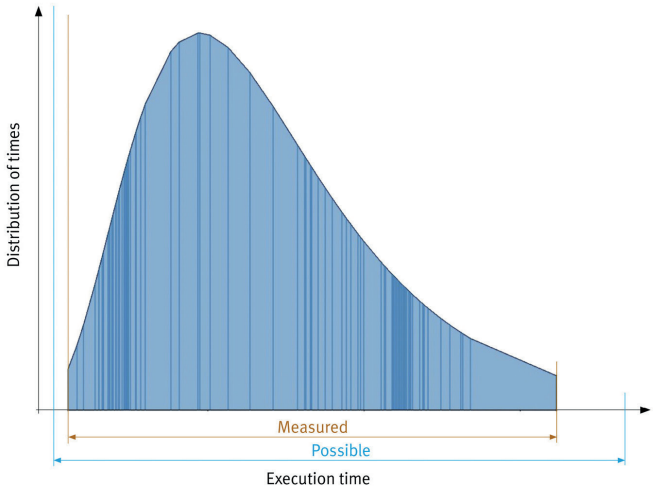
Such a tool automates the measurements, running the specified tests repeatedly and providing a graphical representation of the variation in execution times (Figure 7). It allows the user to select their preferred hardware stress utility (stress-ng [18], for example) to load the different shared resources on the target processor to varying degrees.

In the example, interference can be seen to have a detrimental effect on execution time. The observed WCET is just over 450 million CPU ticks (up from 308), and the mean execution time is just under 229 million CPU ticks (up from 116). If the coverage objectives have been met and the observed WCET falls within bounds, then the interference mitigation is adequate. If not, then the resulting data provides the information required to further optimize the system.

This facility lends itself to performing interference research, and ultimately to demonstrating that the resulting interference mitigation ensures that WCET verification requirements are met.



TBwcet timing diagram without interference



TBwcet timing diagram with interference

Timing Summary

Number of Tests	100
Best-Case Execution Time	57321953.100000
Worst-Case Execution Time	338863903.400000
Minimal Observed Execution Time	63691059 - Test Case 1 (Rep 1)
Maximal Observed Execution Time	308058094 - Test Case 1 (Rep 2)
Mean Observed Execution Time	115596348

Timing Summary

Number of Tests	100
Best-Case Execution Time	59658020.100000
Worst-Case Execution Time	495044356.400000
Minimal Observed Execution Time	66286689 - Test Case 1 (Rep 63)
Maximal Observed Execution Time	450040324 - Test Case 1 (Rep 38)
Mean Observed Execution Time	228836793

Figure 7: Histograms showing the spread of execution times as displayed by the LDRA tool suite.

Interference research and iterative development

Despite this sound foundation, the empirical nature of verification and validation in the MCP environment as compared to single core makes it imperative that the project managers are equipped to adapt readily to changing requirements and configurations.

Dan Iorga et al. [19] recommend a slow and steady approach to measuring and tuning interference. It is highly likely that such interference research will lead to changes in system or software requirements, and conversely that changes in system functional requirements will drive new interference channels or affect existing ones. (Figure 8).

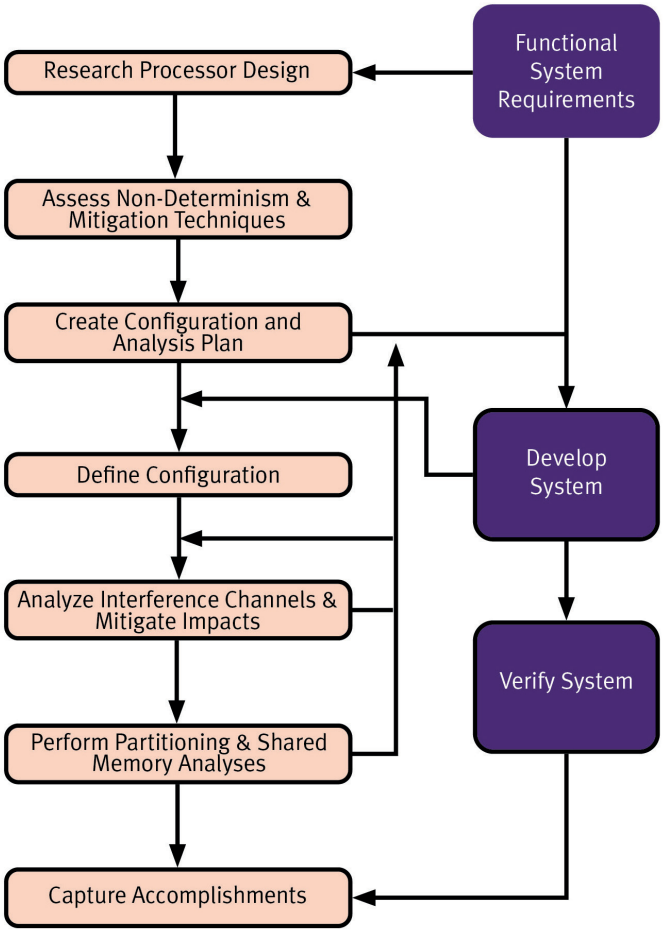


Figure 8: Recommended multicore certification approach [19]

Under these circumstances, an automated mechanism is vital to keep track of what needs revisiting for renewed verification and validation, and hence to keep the project on schedule, and within budget (Figure 9).

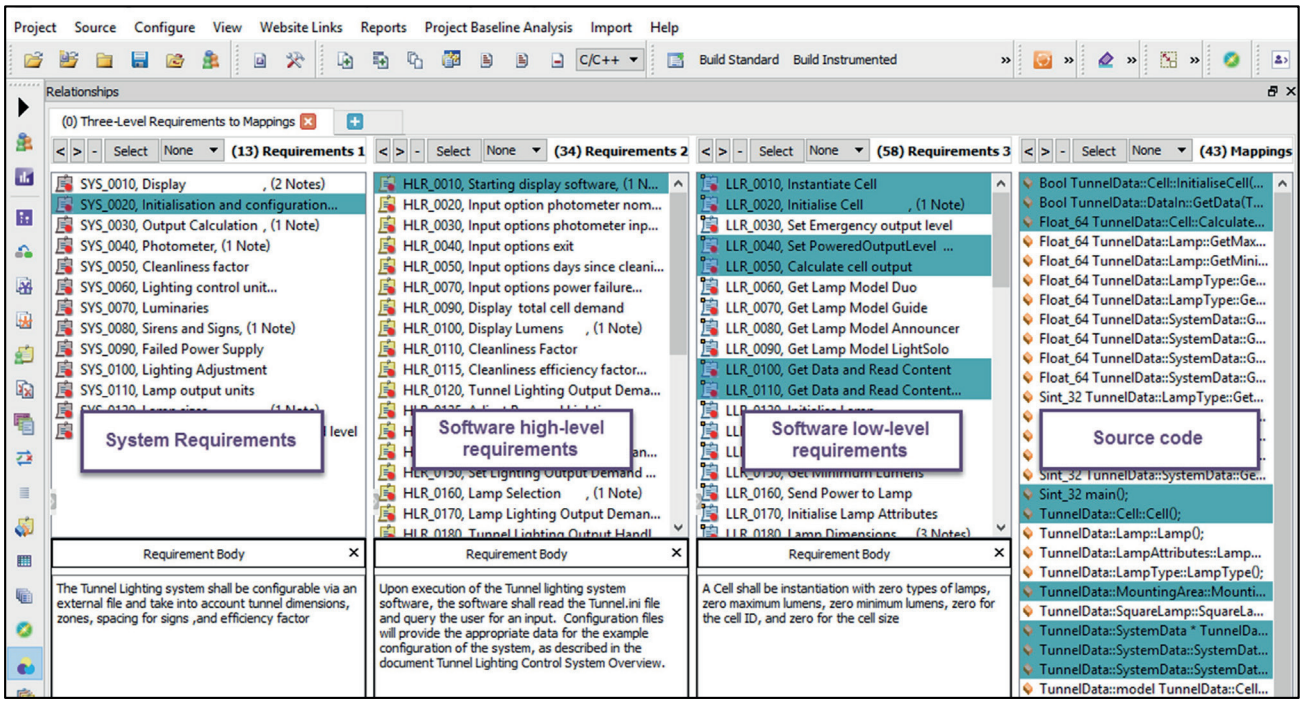


Figure 9: Automating the tracing of requirements and regulatory objectives using the TBmanager [20] component of the LDRA tool suite.

Robust partitioning

Although this iterative approach provides a mechanism to deal with interference from first principles, the use of an RTOS or Hypervisor that offers robust resource and time partitioning is likely to be helpful. In the civil aviation sector, CAST-32A/A(M)C 20-193 makes specific allowance for that:

"*Applicants who have verified that their MCP Platform provides both Robust Resource and Time Partitioning … may verify applications separately on the MCP and determine their WCETs separately*."

That said, the robust partitioning offered by an RTOS, Separation Kernel, or Hypervisor is only as effective as its configuration. Robust partitioning can eliminate many potential interference channels, but the onus remains on the developer to demonstrate that interference mitigation is effective.

Critical real-time multicore applications: other considerations

Aside from WCET, the use of multicore processors impacts several other practices promoted by functional safety standards.

Evidence of compliance

It is important to remember that in the development of a safety-critical system, the considerations given to multicore are merely supplemental to that of a functional safety related process. For example, systems achieving CAST-32A or A(M)C 20-193 compliance are also likely to meet the requirements of DO-178C.

Automating bidirectional traceability to both project requirements (Figure 9) and to the objectives of one or more process standards and guidance documents can help to address a major project management pain point.

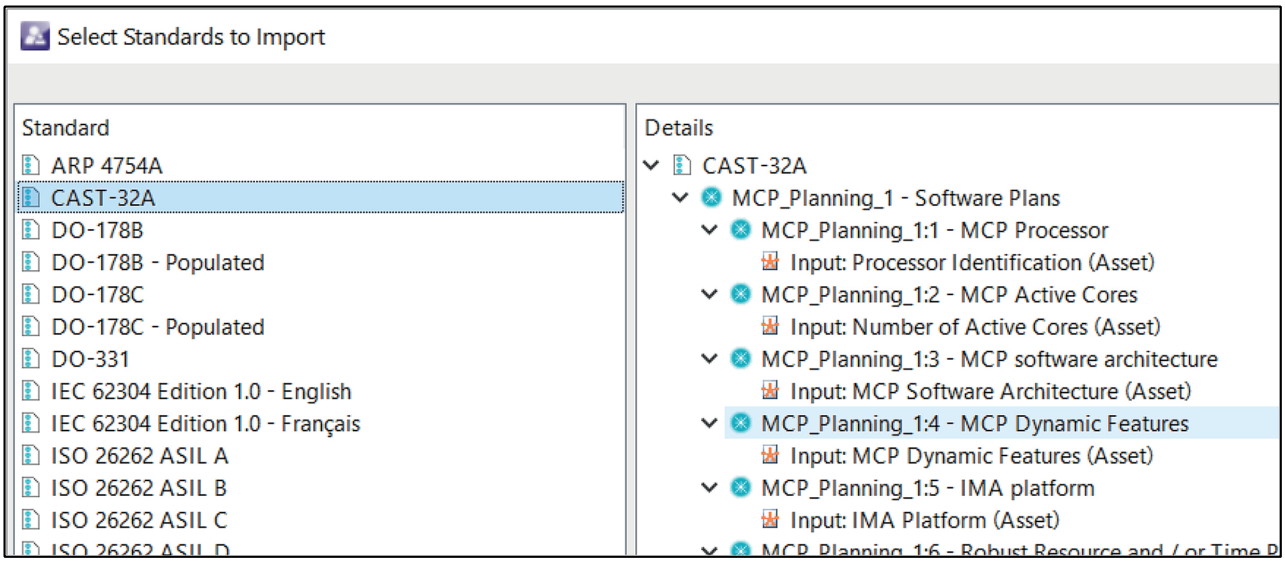


Figure 10: Automating traceability to CAST-32A and A(M)C 20-193 objectives with the TBmanager component of the LDRA tool suite.

Coding standard compliance

Most functional safety standards promote the use of coding standards (or language subsets) such as those promoted by the MISRA Consortium [21]. These are significant in the context of multicore processor-based systems because the rules promoted by these standards guard against the insertions of runtime defects that have the potential to compromise shared resources.

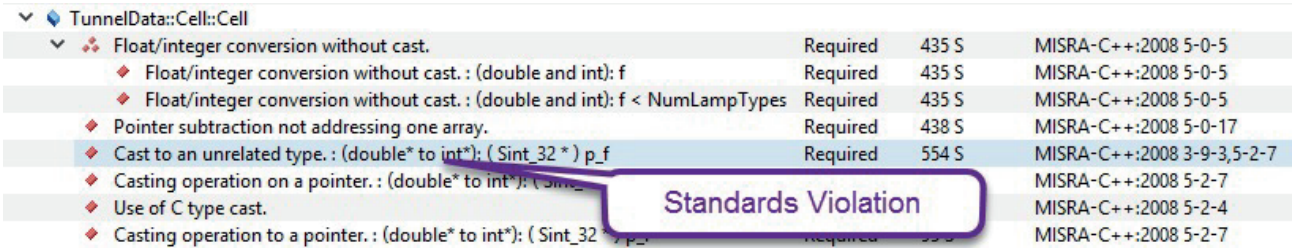


Figure 11: Checking coding standards compliance with the TBvision component of the LDRA tool suite.

Data coupling analysis and control coupling analysis

Software issues such as flawed coupling can introduce delays and variation in execution times, adding to the variation of execution times.

CAST-32A and A(M)C-193 § MCP_Software_2 provides aviation-specific advice that is equally valid in other safety critical sectors in this regard. It states that:

"*The applicant has verified that the data and control coupling between all the individual software components hosted on the same core or on different cores of the MCP has been exercised during software requirement-based testing, including exercising any interfaces between the applications via shared memory and any mechanisms to control the access to shared memory, and that the data and control coupling is correct.*"

Static analysis tools support control coupling and data coupling analysis relating to explicit data and control flow between software components and applications.

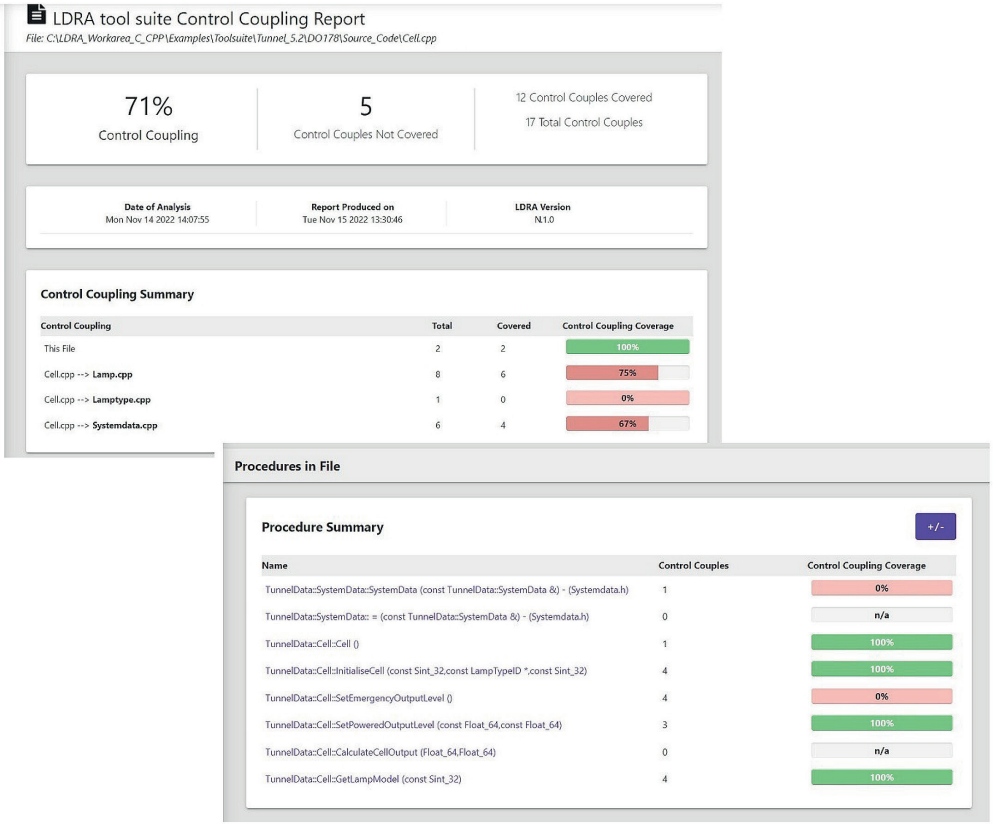


Figure 12: Data coupling analysis and control coupling analysis [22] with the LDRA tool suite

The guidance in the documents also references the possibility of coupling on a platform level, and the WCET analysis techniques discussed earlier relate to that. In most cases control coupling is not a significant factor in multi-core environments. Data coupling, however, is – as the cores need to communicate. Therefore, the emphasis should be on establish and reviewing requirements based tests to exercise any functionality in which data is shared across processors.

Efficient instrumentation for structural coverage analysis on multicore processors

DO-178, IEC 61508, ISO 26262 and other standards concerned with functional safety require structural coverage analysis [23]. Structural coverage data is collated during requirements-based test procedures. The execution of an "instrumented" copy of the source code records the paths taken during execution, and the resulting output submitted for analysis.

Coverage analysis at the software level requires instrumentation probes within the code to track execution. That inevitably has an impact on both performance and resources.

It is necessary to minimize that impact for instrumentation to be effective in a multicore system. One approach to achieving that is to adopt instrumentation techniques that rely on probes inserted into each basic block, meaning that there are as few of them as possible (Figure 13).

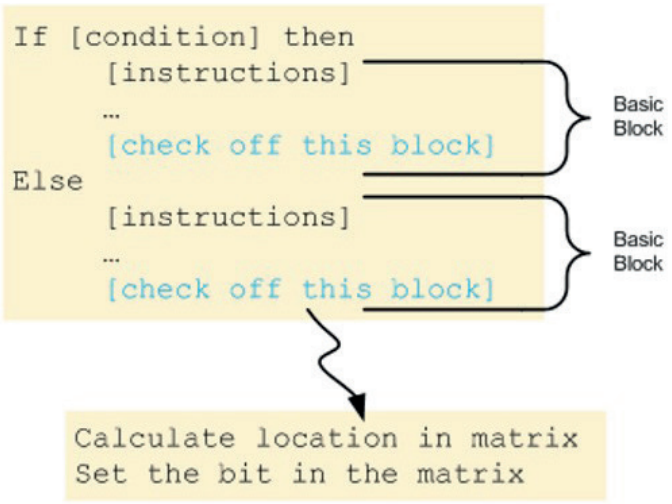


Figure 13: LDRA's instrumentation uses probes that are inserted only into Basic Blocks

With the number of probes minimized, it is also important to the memory required to store the coverage data. Bit-packed storage uses individual bits in a word that correspond to the probe location. That location is pre-calculated, saving execution time (Figure 14).

```
((int)(bitmapstruct.element0 |= (1 << 9)))
```

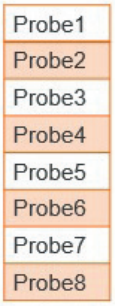


Figure 14: Bit-packed coverage data storage

In a multicore system, collisions are possible. If more than one core writes to the same location, all but the first attempt will fail. However, embedded systems generally run in a loop. It is therefore highly likely that probes will execute repeatedly, and coverage that is unrecorded due to a collision will be recorded on a subsequent pass. Even failing that, the coverage reported is failsafe, because it can never overstate the amount of code exercised.

The alternative approach, involving semaphores or mutexes, would have a significant performance overhead. These remain necessary for MC/DC analysis, where atomic locking ensures the accuracy and completeness of results.

Conclusions

Concerns over the suitability of Multicore Processors (MCPs) for safety critical applications are nothing new. But the relatively low volume of the safety critical sectors has seen a situation evolve where MCPs based critical applications have deployed a single core only. In a world where other applications are seeing huge benefits from the SWaP advantages of fully utilized MCPs, that cannot be an ideal situation.

However, there are challenges with MCPs in these environments. The use of a symmetric multiprocessing operating system is not a viable option, and robust partitioning between domains is critical. However that is achieved – via an RTOS, separation kernel, or hypervisor – hardware interference represents another challenge.

Even for a single core, the calculation of Worst-Case Execution Times (WCET) through static analysis alone can only ever be an approximation. Combining that approximate approach with the additional demands of MCPs where these approximations are apparent across all cores is clearly a suboptimal approach. It is far better to determine where the most demanding execution paths lie, and then measure their duration dynamically.

One recommendation is the use of an iterative development model to overcome this challenge, repeatedly testing and adapting the system as it is developed to ensure that the potential for such interference is minimized. In sophisticated projects, keeping track of the fulfilment of standard objectives and project requirements can be difficult enough even without the introduction of such a process. An integrated, automated requirements traceability system is invaluable.

In summary, there are real challenges in the application of MCPs in the safety critical sector, but none that are insurmountable given the right tools, used diligently.

Works cited

[1] M. Kyrnin, "Multiple Core Processors: Is More Always Better?," 28 July 2020. [Online]. Available: https://www.lifewire.com/multiple-core-processors-832453. [Accessed 1 November 2021].

[2] NVIDIA Corporation, "The Benefits of Multiple CPU Cores in Mobile Devices," 2010. [Online]. Available: https://www.nvidia.co.uk/content/PDF/tegra_white_papers/Benefits-of-Multi-core-CPUs-in-Mobile-Devices_Ver1.2.pdf. [Accessed 1 November 2021].

[3] R. W. e. al., "The Worst-Case Execution-Time Problem—Overview of Methods and Survey of Tools," April 2008. [Online]. Available: http://www.es.mdh.se/pdf_publications/1258.pdf. [Accessed 1 November 2021].

[4] C. L. LIU and JAMES W. LAYLAND, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," Journal of the Association for Computing Machinery, vol. 20, no. 1, pp. 46-61, January 1973.

[5] T. Loveless, "Challenges building safe multicore systems," 15 June 2020. [Online]. Available: https://www.lynx.com/embedded-systems-learning-center/challenges-building-safe-multicore-mcp-software-systems. [Accessed 2 November 2021].

[6] Certification Authorities Software Team (CAST), Position Paper CAST-32A - Multicore processors, CAST, 2016.

[7] RTCC, DO-178C "Software Considerations in Airbourne Systems and Equipment Certification", RTCA, 2011.

[8] EASA, "AMC 20-193 Use of multi-core processors," 2022. [Online]. Available: https://www.easa.europa.eu/sites/default/files/dfu/annex_i_to_ed_decision_2022-001-r_amc_20-193_use_of_multi-core_processors_mcps.pdf. [Accessed 18th January 2023].

[9] International Electrotechnical Commission (IEC), IEC 61508:2010 “Functional safety of electrical/electronic/programmable electronic safety-related systems” (all parts), IEC, 2010.

[10] International Organization for Standardization, ISO 26262:2018 “Road vehicles - Functional safety”, International Organization for Standardization, 2018.

[11] European Committee for Electrotechnical Standardization (CENELEC), EN 50128 “Railway applications - Communication, signalling and processing systems - Software for railway control and protection systems”, CENELEC, 2020.

[12] International Electrotechnical Commission, IEC 62304:2006+AMD1 Medical device software - Software life cycle processes, IEC, 2015.

[13] U S Food & Drug Administration, “Classify your medical device,” 2 July 2020. [Online]. Available: https://www.fda.gov/medical-devices/overview-device-regulation/classify-your-medical-device. [Accessed 30 December 2022].

[14] Halstead, Maurice H., Elements of Software Science., Amsterdam: Elsevier North-Holland, 1977.

[15] LDRA, “LDRA Testbed® and TBvision®,” LDRA, [Online]. Available: https://ldra.com/products/ldra-testbed-tbvision/. [Accessed 23 March 2022].

[16] LDRA, “TBrun®,” LDRA, [Online]. Available: https://ldra.com/products/tbrun/. [Accessed 23 March 2022].

[17] LDRA, “TBwcet®,” [Online]. Available: https://ldra.com/products/tbwcet/. [Accessed 3rd February 2023].

[18] C. King, “Ubuntu wiki: stress-ng,,” 7th October 3030. [Online]. Available: https://wiki.ubuntu.com/Kernel/Reference/stress-ng. [Accessed 3rd February 2023].

[19] Paul J. Parkinson (Wind River) and Harold G. Tiedeman (Rockwell Collins), “Plan With Confidence: Route to a Successful DO-178C Multi-Core Certification,” 28 September 2018. [Online]. Available: https://www.slideshare.net/ICTperspectives/plan-with-confidence-route-to-a-successful-do178c-multicore-certification. [Accessed 5 November 2021].

[20] LDRA, “TBmanager®,” [Online]. Available: https://ldra.com/products/tbmanager/. [Accessed 3rd February 2023].

[21] The MISRA Consortium Limited, “The MISRA website,” The MISRA Consortium Limited, 2021. [Online]. Available: https://www.misra.org.uk/. [Accessed 13 April 2022].

[22] LDRA, “Control coupling analysis and data coupling analysis for embedded software,” [Online]. Available: https://ldra.com/capabilities/data-couplingcontrol-coupling/. [Accessed 3rd February 2023].

[23] LDRA, “Structural (Code) Coverage Analysis in embedded systems,” [Online]. Available: https://ldra.com/capabilities/code-coverage-analysis/. [Accessed 3rd February 2023].

LDRA UK & Worldwide
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology Inc.
2540 King Arthur Blvd, 3rd Floor, 12th Main, Lewisville, Texas 75056
Tel: +1 (855) 855 5372
e-mail: info@ldra.com

LDRA Technology Pvt. Ltd.
Unit B-3, Third floor Tower B, Golden Enclave
HAL Airport Road Bengaluru 560017
Tel: +91 80 4080 8707
e-mail: india@ldra.com