LDRA
Software Technology

Technical White Paper

Following the recommendations of CAST-32A & A(M)C 20-193

www.ldra.com

Contents

Introduction

The Certification Authorities Software Team [1] (CAST) were a group of civil aviation regulatory officials from Asia, Europe, North and South America, including the FAA [2] and EASA [3]. CAST position papers are for the clarification of issues that related to the formally established standards in that sector. They covered a range of topics, including the clarification of aspects of the standards that have proven ambiguous, and the handling of new technologies. The latter is clearly the case with CAST-32A "Multi-core Processors" [4], the last position paper to be produced by the team. It will be deprecated when its successor documents AC 20-193 [5] and AMC 20-193 (known collectively as A(M)C 20-193 [6]) have both been published.

CAST papers are authoritative, and their advice is often adopted even before it is formally integrated into subsequent published standards. However, neither CAST-32A nor A(M)C 20 193 is prescriptive. They are intended to guide and support the developer in their quest to meet a standard - typically DO-178C [7].

Most of us are familiar with the benefits that multicore processors (MCPs) have brought to our daily lives. After all, they have been available in personal computers since the early 2000s [8], and NVIDIA was promoting their benefits in mobile devices as long ago as 2010 [9]. So, what made the CAST team turn their attention to them?

Critical applications and MCPs

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) MCPs.

These processors are built for SMP (Symmetric MultiProcessing) operating systems. These operating systems schedule processes across the cores 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 civil aviation control 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 often a simple matter of increasing the speed of the processor.

Real-time control systems are closed loop, allowing only specified time windows to gather data, process that data, and update the system.
"Soft" real-time control systems might be deployed by an in-flight entertainment system, where the occasional missed deadline is not a big deal. Conversely, a missed deadline for a "hard" real-time control typified by flight control systems can be catastrophic.

Processors and processing

Following the early beginnings of simple homogenous MCPs and SMP OSs, several variants have emerged.

Homogeneous (symmetric) MCP
Cores are all the same type.

Heterogeneous (asymmetric) MCP
Uses a mix of core types. Some of those may be specialised cores (for example, such as graphics processing units).

Asymmetric multiprocessing (AMP)
A separate OS, or a separate instantiation of the same OS, runs on each CPU.

Symmetric multiprocessing (SMP)
A single instantiation of an OS manages all CPUs simultaneously, and applications can float to any of them.

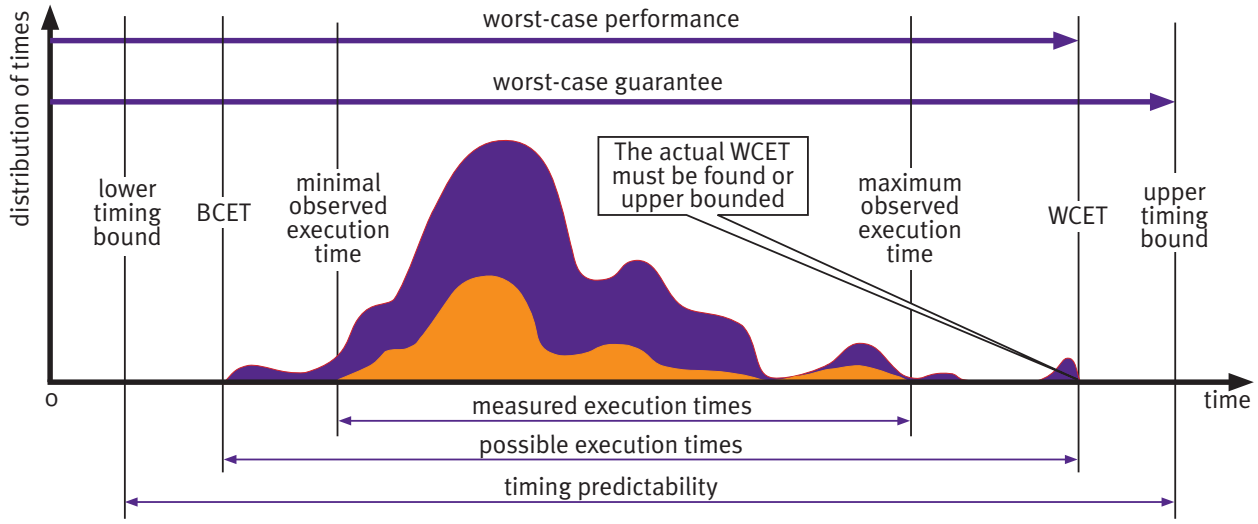
Bound multiprocessing (BMP)
A single instantiation of an OS manages all CPUs simultaneously, but each application is locked to a specific CPU.

Hardware and software architectures have also evolved, potentially impacting the calculation and management of those time windows still further (sidebar [10] [11]).

The criticality of worst-case execution times

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 and (hence) ensure deterministic behaviour. Unfortunately, it is not possible in general to determine worst-case execution times for programs.

Suppose that a real-time system runs on a single-core processor and consists of several tasks which realize the required functionality. Figure 1 depicts several relevant properties of a real-time task [12].



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

Figure 1: 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

Within the mechanics of an operating system and in the context of this paper, suppose that a process is a task scheduled by an operating system.

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 [13], 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 deadlines has no efficient algorithm.

The operating systems used in laptops and mobile phones do good jobs, 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 2). For example, often an entire hierarchical memory is shared so that interference is possible in many places.

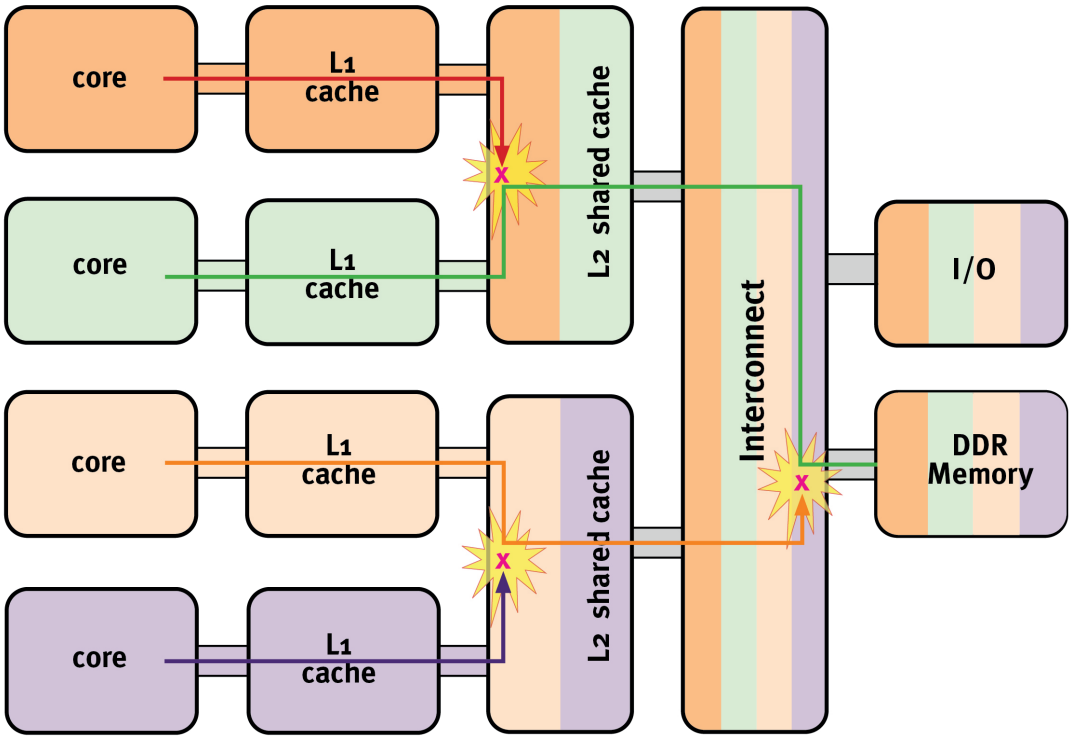


Figure 2: A representation of hardware interference in shared hierarchical memory [14]

These interference channels cause the execution time distribution to spread. Instead of a tight peak, the distribution becomes right skewed with a long tail (Figure 3) [14].

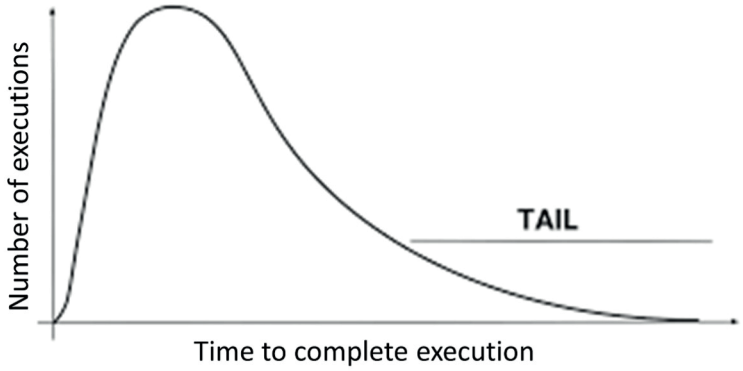


Figure 3: Execution time distribution is skewed by the effects of interference.

It doesn't stop with memory. For example, Hardware Shared Resources (HSRs) on the Xilinx Zynq UltraScale+ [15] with its ARM Cortex-A53 CPU cores include:

- Snoop control unit
- Accelerator coherency port
- L2 cache
- Cache coherent interconnect
- Central switch
- Direct memory access controller
- DDR RAM
- Bus interface unit
- On-chip interconnect
- Translation control unit

Outside the realm of safety critical applications, these issues are of little consequence. But where functional safety is paramount, they are critical. CAST-32A and A(M)C 20-193 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."

It is therefore clear that the laissez-faire approach that is entirely reasonable for a laptop or a mobile phone cannot apply here.

Dealing with the demands of CAST-32A & A(M)C 20-193

Figure 4 is a representation of the relationships between the key civil aviation documents relevant to this paper. In general, the following discussion references the FAA nomenclature ("DO…") but it is equally applicable to the EASA harmonized documents ("ED…") also shown in the illustration.

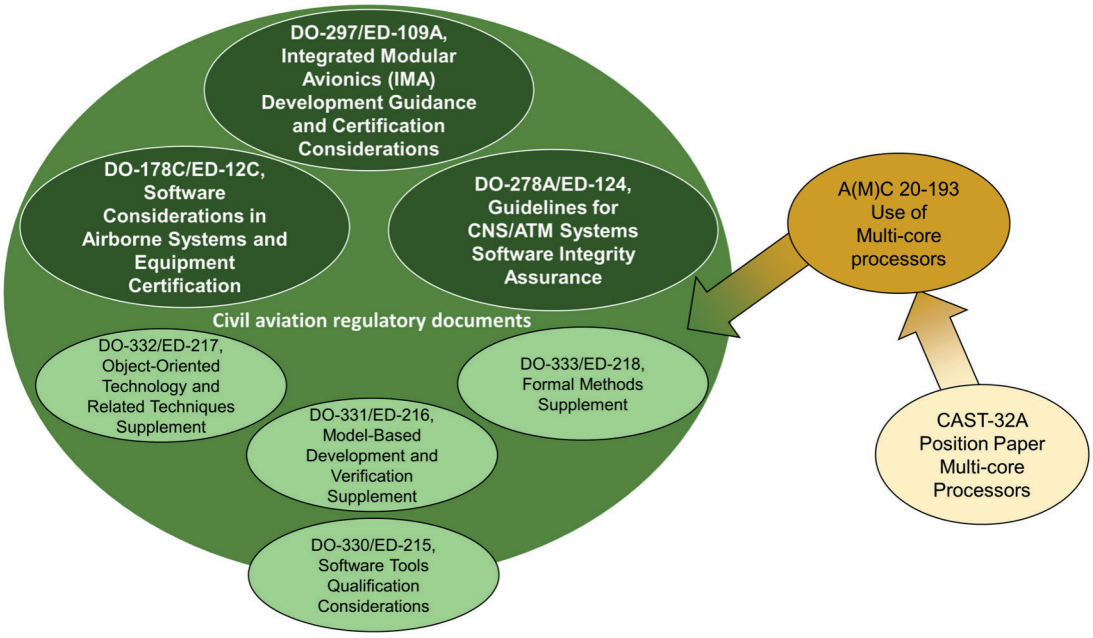


Figure 4: A representation of civil aviation regulatory and advisory docoments[1]

As a result of the issues surrounding execution times and MCPs, CAST-32A & A(M)C 20 193 place significant focus on the provision of evidence that allocated resources are adequate to allow for worst case execution times.

Adapting the development process to enable such provision requires a mechanism to analyse execution times, and a controlled, iterative development process to allow for their optimisation.

The analysis of execution times

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.

[1]CNS/ATM is an abbreviation for Communication, Navigation, Surveillance, and Air Traffic Management. Also note that DO 330 is shown to be overlapping the civil aerospace sector because it is also applicable outside that domain.

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 [12]. 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 [16] 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 5). 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 5: Halstead's Metrics calculated using the LDRA tool suite.

The same static analysis also yields call diagrams associated with the code base, presenting a means of visualizing where the most demanding functions revealed by this analysis are exercised in the context of the code base (Figure 6).

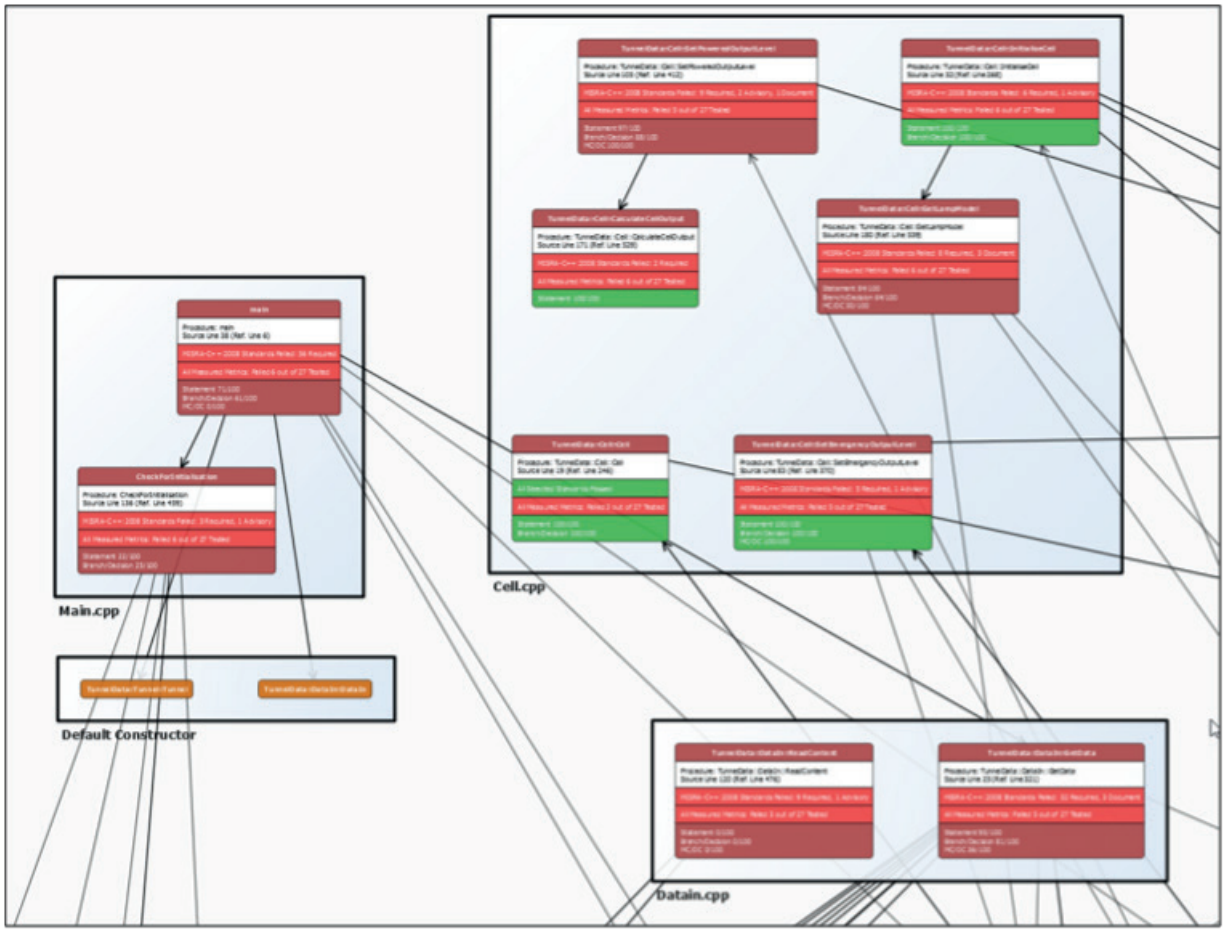


Figure 6: Call diagrams in the LDRA tool suite provide a means to visualize where the most demanding functions are called.

The only way to ascertain how that information translates into time elapsed for a particular function or call tree is 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 7).

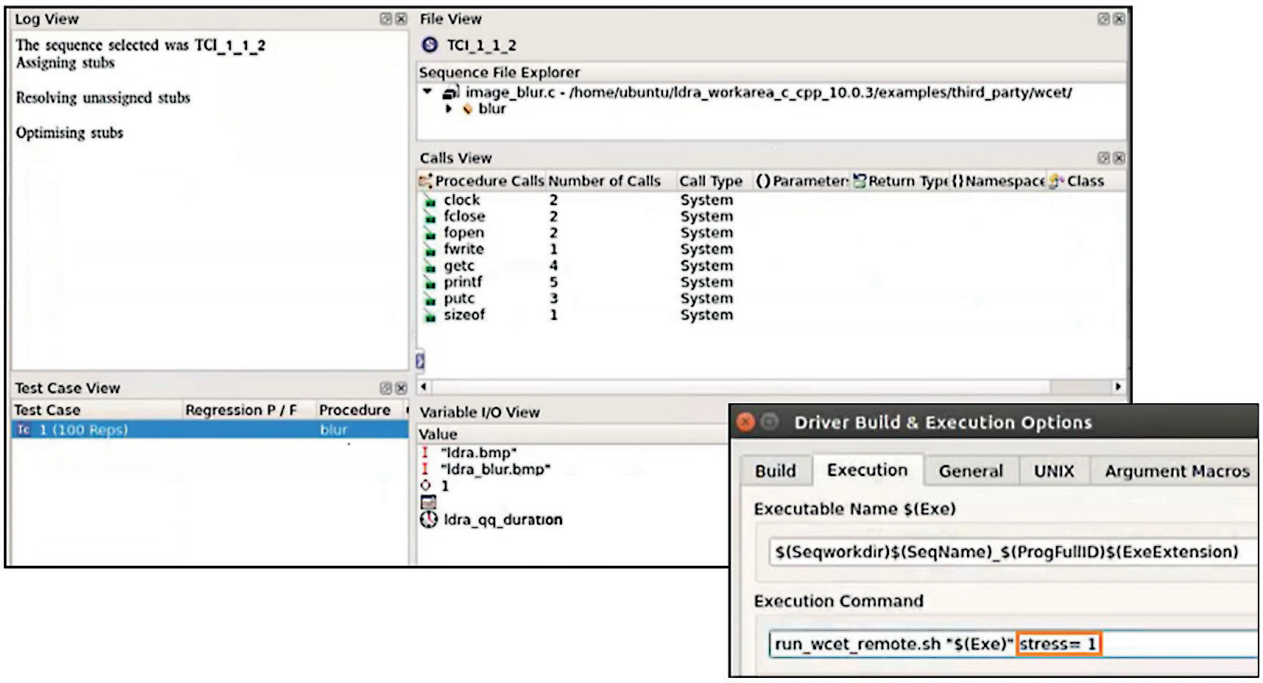


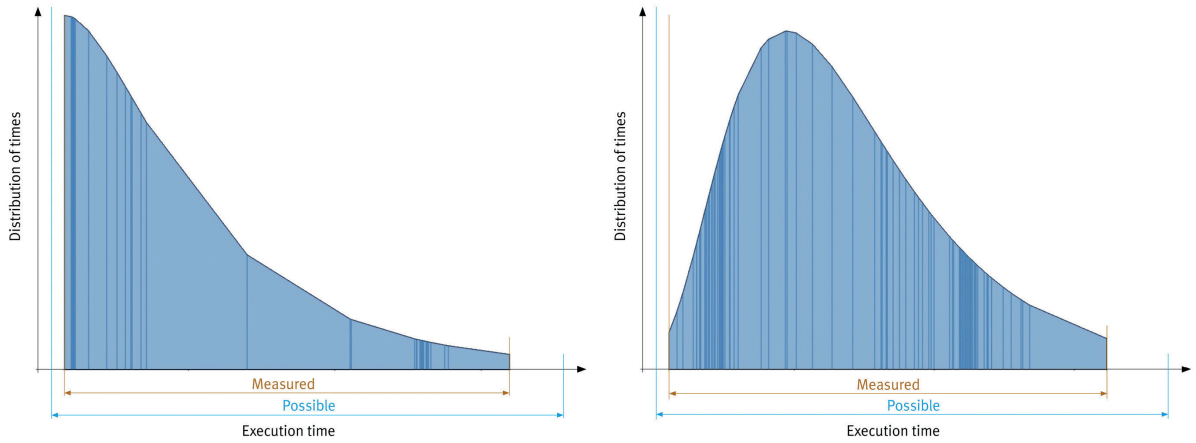
Figure 7: Using the TBrun [17] component of the LDRA tool suite with the TBwcet [18] module to measure execution times.

This tool automates the measurements, running the specified tests repeatedly and providing a graphical representation of the variation in execution times (Figure 8). It allows the user to select their preferred hardware stress utility (stress-ng [19], for example) to load the different shared resources on the target processor to varying degrees.

The example shows that interference has had 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 are 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.

It is also important to analyse the performance of the application of a whole. This is achieved by leveraging the LDRA tool suite's source substitution mechanism to integrate those unit test driver(s) into the application.

TBwcet provides the flexibility to perform system tests or to drill down into performance at a unit test level. It therefore presents the ideal toolkit for performing interference research, and ultimately for demonstrating that WCET verification requirements are met.



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

Embracing interference research

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

Dan Iorga et al. [20] recommend a slow and steady approach to measuring the effects of interference and mitigating for them. The LDRA tool suite is ideally placed to support such interference research, which will likely lead to changes in system or software requirements. Conversely, changes in system functional requirements will also drive new interference channels or affect existing ones (Figure 8).

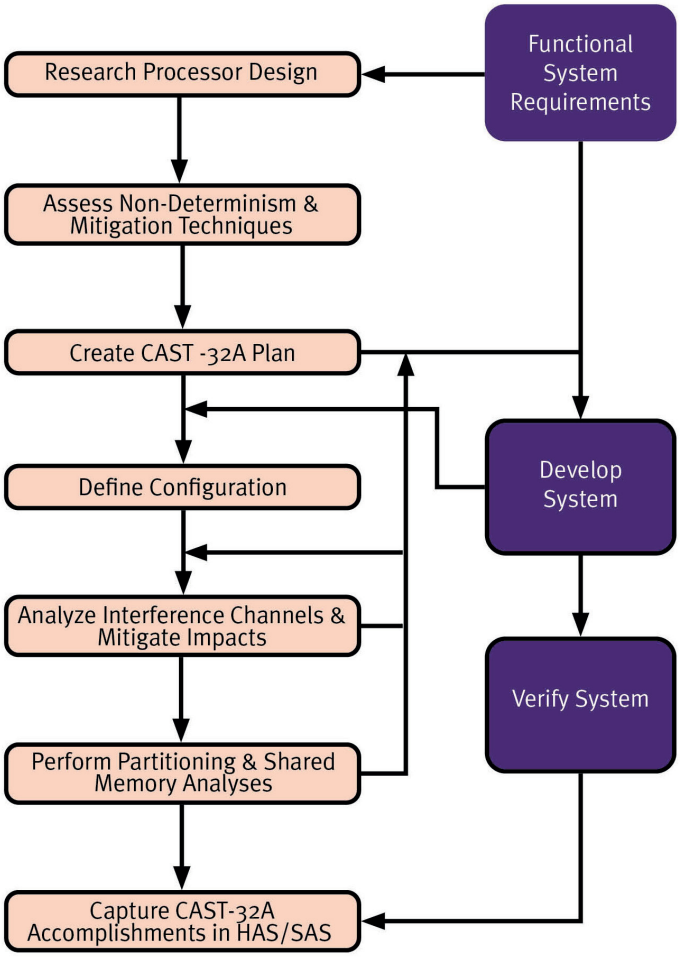


Figure 9: Recommended multicore certification approach [20]

Under these circumstances, an automated mechanism is helpful 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).

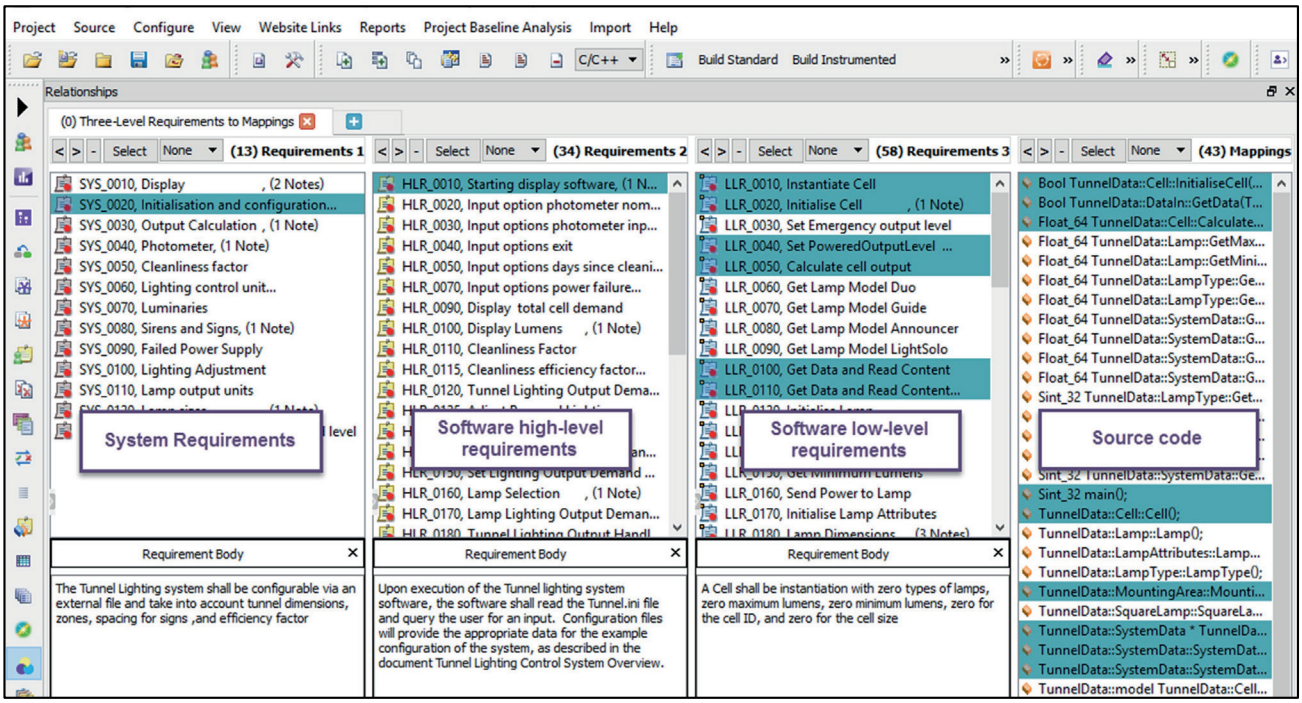


Figure 10: Tracing requirements and objectives with the TBmanager component of the LDRA tool suite

CAST-32A & A(M)C 20-193 Objectives

There are ten objectives in CAST-32A & A(M)C 20-193 which supplement the DO-178C processes for multicore processors throughout its V-model lifecycle. They span from planning through to verification, while focusing on shared resources and interference channels. Neither CAST 32A nor A(M)C 20-193 prescribe methods for achieving those objectives, leaving that to the system integrator and their suppliers.

In general, because multicore interference is inherently a hardware issue, there are several helpful options implemented in software that interacts directly with that hardware.

Traditionally that interface consists of a RTOS pre-certified to DO-178C. These are marketed by specialist vendors including DDC-I [21], Green Hills [22], and Lynx [23] to name just a few. That is no longer the only option with some of these vendors now offering separation kernel and hypervisor-based solutions that are purported to lend themselves to certification more readily [24], [25]. Irrespective of the form this host software takes – RTOS, Separation Kernel, or whatever - if it manages even some shared resources and ensures domain separation, then it will help to achieve compliance with CAST-32A and A(M)C 20-193.

The objectives laid out by the CAST-32A & A(M)C 20-193 objectives detail the issues surrounding the adoption of these techniques. Key considerations include the following.

MCP_Planning_1

"*The applicant's software plans or other deliverable documents…*"

The first part of CAST-32A & A(M)C 20-193 relates to the planning process. It discusses the specific items to be considered in planning a project around multicore execution, such as the device to be used, the number of active cores to be deployed, dynamic features in the MCP and so on, with a particular focus on with respect to resource management. It talks about how the processor is architected, what components will potentially lead to contention during verification and validation processes, what issues there are to be aware of, and the identification of appropriate tools.

Automated tracking of the concurrent fulfilment of CAST-32A & A(M)C 20-193 objectives, those of a standard (typically DO-178C), and the project specific requirements can help to address a project management headache (Figure 10).

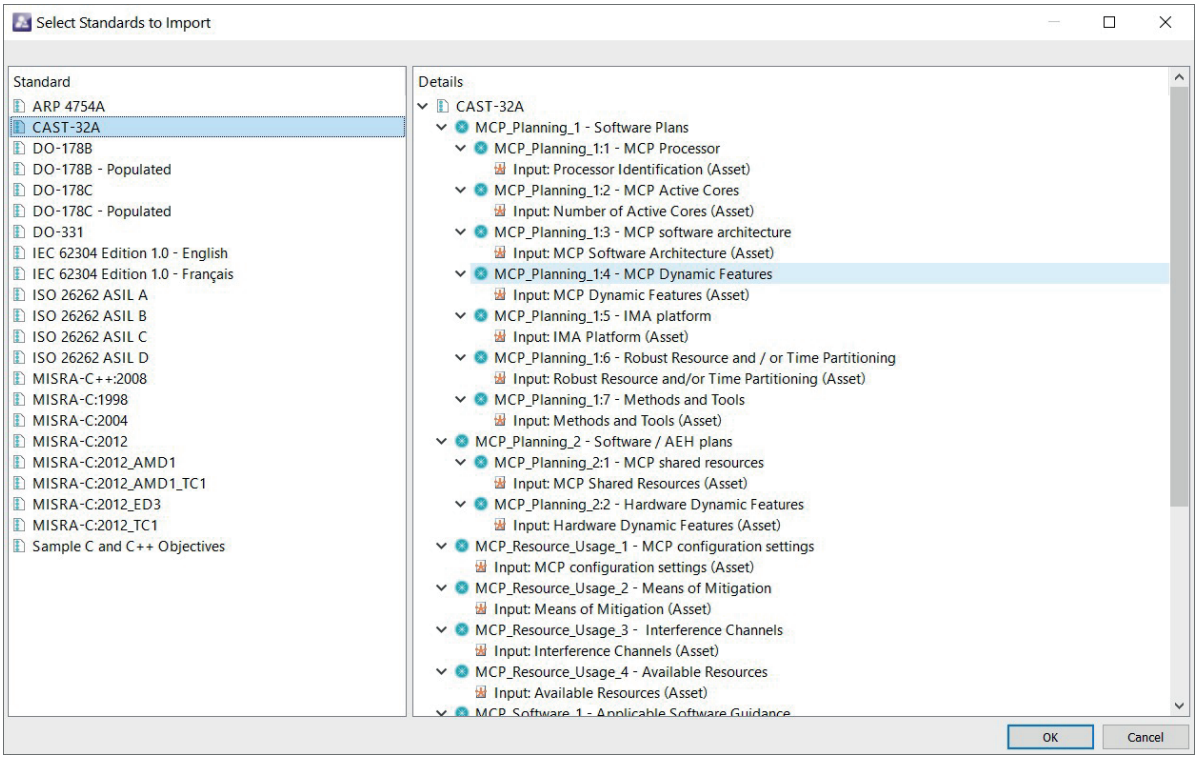


Figure 11: Importing CAST-32A & A(M)C 20-193 objectives into the TBmanager component of the LDRA tool suite

MCP_Resource_Usage_1

"The applicant has determined and documented the MCP configuration settings…"

MCPs are extremely complex and have many settings that can affect execution and potentially impact system safety. It is important to consider the desired multicore processor usage patterns, ensuring that all configuration bits are configured correctly such that timing, execution, and memory access work in harmony. As previously discussed, the empirical nature of MCP configuration and scheduling optimization makes it highly likely that the configuration will change during development. Keeping track of changing statuses when it does so is important.

Objectives	Artifacts	Assets	Last Verifi...	Objective ...	Objective St...	Objective S...	Placeholders
MCP_Accomplishment_Summary_1	0	1	Thu Sep 30 ...	Accomplish...	CAST-32A	Partial	100.00%
MCP_Error_Handling_1	0	1	Thu Sep 30 ...	Failures	CAST-32A	Partial	100.00%
MCP_Planning_1	0	0	Thu Sep 30 ...	Software Pl...	CAST-32A	Partial	0.00%
MCP_Planning_1:1	0	1	Thu Sep 30 ...	MCP Proces...	CAST-32A	Partial	100.00%
MCP_Planning_1:2	0	1	Thu Sep 30 ...	MCP Active...	CAST-32A	Partial	100.00%
MCP_Planning_1:3	0	1	Thu Sep 30 ...	MCP softwa...	CAST-32A	Partial	100.00%
MCP_Planning_1:4	0	1	Thu Sep 30 ...	MCP Dyna...	CAST-32A	Partial	100.00%
MCP_Planning_1:5	0	1	Thu Sep 30 ...	IMA platform	CAST-32A	Partial	100.00%
MCP_Planning_1:6	0	1	Thu Sep 30 ...	Robust Res...	CAST-32A	Partial	100.00%
MCP_Planning_1:7	0	1	Thu Sep 30 ...	Methods an...	CAST-32A	Partial	100.00%
MCP_Planning_2	0	0	Thu Sep 30 ...	Software / ...	CAST-32A	Partial	0.00%
MCP_Planning_2:1	0	1	Thu Sep 30 ...	MCP shared...	CAST-32A	Partial	100.00%
MCP_Planning_2:2	0	1	Thu Sep 30 ...	Hardware ...	CAST-32A	Partial	100.00%
MCP_Resource_Usage_1	0	1	Thu Sep 30 ...	MCP config...	CAST-32A	Partial	100.00%
MCP_Resource_Usage_2	0	1	Thu Sep 30 ...	Means of M...	CAST-32A	Partial	100.00%
MCP_Resource_Usage_3	0	1	Thu Sep 30 ...	Interferenc...	CAST-32A	Partial	100.00%
MCP_Resource_Usage_4	0	0	None	Available R...	CAST-32A	Unfulfilled	0.00%
MCP_Software_1	0	1	Thu Sep 30 ...	Applicable ...	CAST-32A	Partial	100.00%
MCP_Software_2	0	1	None	Data and C...	CAST-32A	Partial	16.67%

Figure 12: A graphical representation of progress towards meeting CAST-32A & A(M)C 20-193 objectives in the TBmanager component of the LDRA tool suite

MCP_Resource_Usage_2

"*The applicant has … a means that ensures that in the event of the Critical Configuration Settings of the MCP being inadvertently altered, and appropriate means of mitigation is required*".

This objective is focused on the documentation of settings that could inadvertently affect the operation of the system – including configuration registers, a clear understanding of which resources are dedicated to a particular core (such as registers) and which are shared (such as shared memory).

The configuration of the MSR register is typical example of a critical configuration setting. This register can set many critical configuration bits such as use of cache, access to memory, and hyperthreading. The use of the latter is expressly discouraged by CAST-32A & A(M)C 20-193 because it adds many non-deterministic effects around resource usage.

MCP_Resource_Usage_3

"*The applicant has identified the interference channels that could permit interference to affect the software applications hosted on the MCP cores, and has verified the applicant's chosen means of mitigation of the interference*".

There are many possible interference channels. The objective here is to identify those likely to cause an issue for the application under development, and to identify a means of mitigation.

MCP_Resource_Usage_4 and MCP_Software_1

The fourth resource usage objective and the first software objective are both concerned with ensuring that that allocated tasks can be achieved in the time allotted to them, and that adequate resources are allocated for that to be achieved.

MCP_Resource_Usage_4

"The applicant has identified the available resources of the MCP and of its interconnect … has allocated the resources of the MCP to the software applications hosted on the MCP and has verified that the demands … do not exceed the available resources…

NOTE: The need to use Worst Case scenarios is implicit in this objective."

MCP_Software_1

"*… the applicant has verified that all the hosted software components function correctly and have sufficient time to complete their execution when all the hosted software is executing in the intended final configuration."*

CAST-32A & A(M)C 20-193 go on to suggest that "*Applicants who have verified that their MCP Platform provides both Robust Resource and Time Partitioning (as defined in this document) may verify applications separately on the MCP and determine their WCETs separately.*"
This implied desirability of robust partitioning echoes the stated position in DO-297, "Integrated Modular Avionics" [26] (sidebar [27]).

As CAST-32A & A(M)C 20-193 imply, the use of robust partitioning is likely to make interference mitigation easier. That said, the robust partitioning offered by an RTOS, Separation Kernel, or Hypervisor is only as effective as its configuration and is unlikely to mitigate against every source of hardware interference.

Whether robust partitioning is in place or not, it is significant that DO-178C places the onus on the developer to demonstrate that interference mitigation is effective – bearing in mind that DO 178C is prescriptive, whereas CAST-32A & A(M)C 20-193 are not.

CAST-32A & A(M)C 20-193, and DO-297 Integrated Modular Avionics (IMA)

DO-297 "Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations" [26] deals with use of powerful computing hardware to host software functions of differing safety criticality through the use of robust partitioning.

A helpful insight into the common issues of partitioning, domain separation, and isolation techniques are discussed at length in the paper "A Framework for Analyzing Shared Resource Interference in a Multicore System".

MCP_Software_2

"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."

Data and control coupling, which are important considerations in DO-178C, are reiterated here with a particular focus on shared memory and shared resources.

LDRA's static analysis tools support control coupling and data coupling analysis relating to explicit data and control flow between software components and applications (Figure 13).

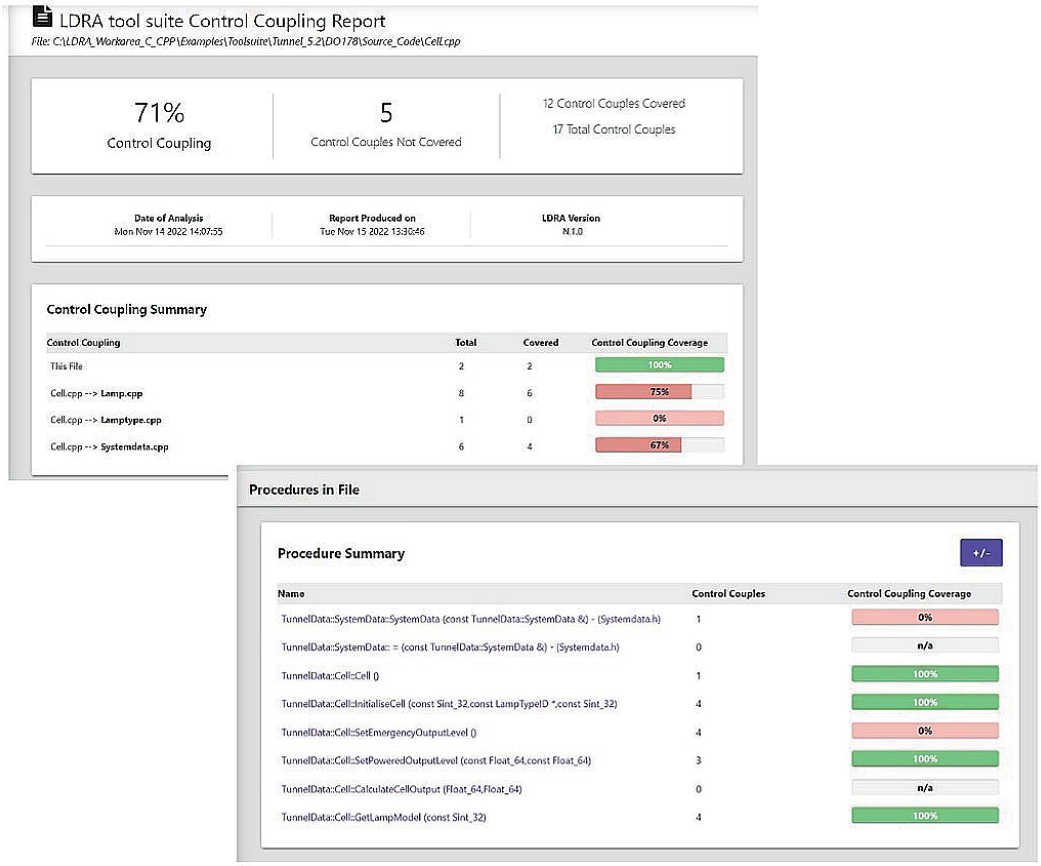


Figure 13: Data coupling analysis and control coupling analysis [28] with the LDRA tool suite

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. To comply with that guidance, the emphasis should be on establishing and reviewing requirements based tests to exercise any functionality in which data is shared across processor cores.

Other V&V considerations for DO-178C compliant MCP applications

Validation and verification in accordance with the appropriate standard (typically DO-178C) needs to include provision to account for the additional risk implied by the MCP environment.

Many of the tools traditionally used in the verification of systems designed for single core processors are equally important in mitigating that risk.

It is useful to place these processes in the context of a development lifecycle model used in DO-178C compliant civil aviation applications. That standard only requires that a lifecycle model is defined and stops short of dictating which one. However, a V-model like that shown in Figure 14 is commonly used [29].

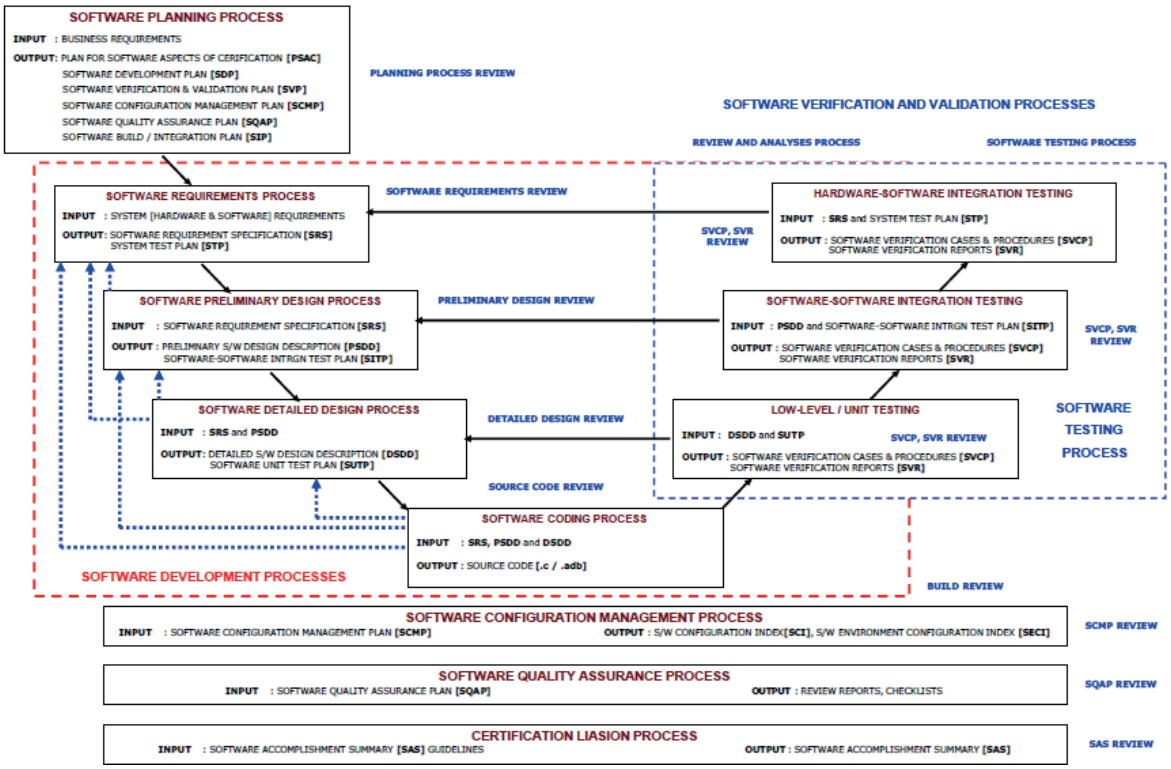


Figure 14: Typical lifecycle model as used in civil aerospace [29]

The contrast of this model with the iterative, evolutionary model illustrated in Figure 9 earlier highlights the challenges from a project management perspective. There are established mechanisms to manage changes due to (for example) changed customer requirements and failed tests. However, the iterative process necessary to leverage MCPs needs to embrace change, which brings a new emphasis to the need to track, control, and automate the mechanisms involved.

The benefits of automated bidirectional traceability have already been discussed, but there are several other situations where an automated, integrated development environment help to ease the path.

DO-178C, code review, and MCPs

The term “static analysis” implies that the source code is “inspected” in some way, rather than some or all of it being compiled and executed.

Code review allows the code to be verified against a coding standard, such as MISRA C [30], MISRA C++ [31], JSF++ [32], CERT C [33], and CERT C++ [34]. That is significant in the context of CAST-32A because the rules promoted by these standards guard against the insertions of runtime defects that have the potential to compromise shared resources.

Similarly, static analysis tools can also ensure the absence of recursion within the code base, avoid unbounded loops, and measure the stack depth.

It is important that any code changes necessary to resolve any emerging interference issues maintain those same standards throughout, to ensure that modifications to cure one problem do not inadvertently introduce others. Automated traceability helps ensure that any new code is highlighted and rechecked for conformance. (Figure 15).

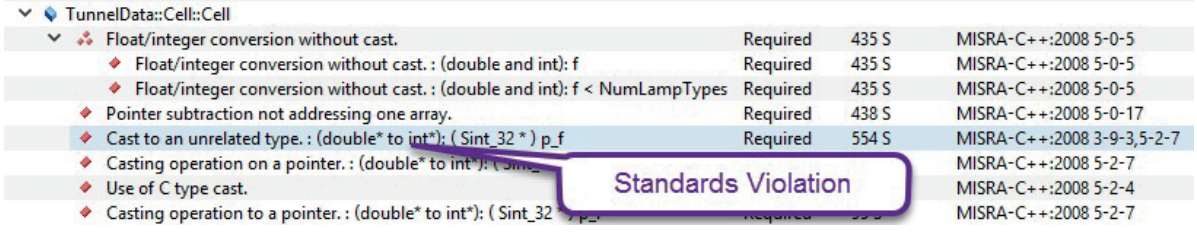


Figure 15: Checking for compliance with the MISRA C++:2008 coding standard using the LDRA tool suite

DO-178C, structural coverage analysis, and MCPs

Dynamic analysis, like static analysis, is leveraged in many ways during the development of any standards-compliant civil avionics system.

Dynamic analysis involves the execution of some or all the code base. It therefore encompasses unit (or low-level), integration and system level analysis. Within the LDRA tool suite, coverage data can be combined from these different techniques.

Like static analysis, where the iterative nature of MCP system development requires code to be restructured, that requirement is automatically highlighted in the TBmanager component.

Structural Coverage (SC) data is collated during requirements-based test procedures. A copy of the source code is typically "instrumented" with function calls to show the paths taken during execution, and the resulting output submitted to Structural Coverage Analysis (SCA). Unexecuted sections of the code may require changes to test cases or requirements, or removal of dead code (Figure 16). An iterative "review, analyse, verify" cycle is typically needed to ensure that objectives are met.

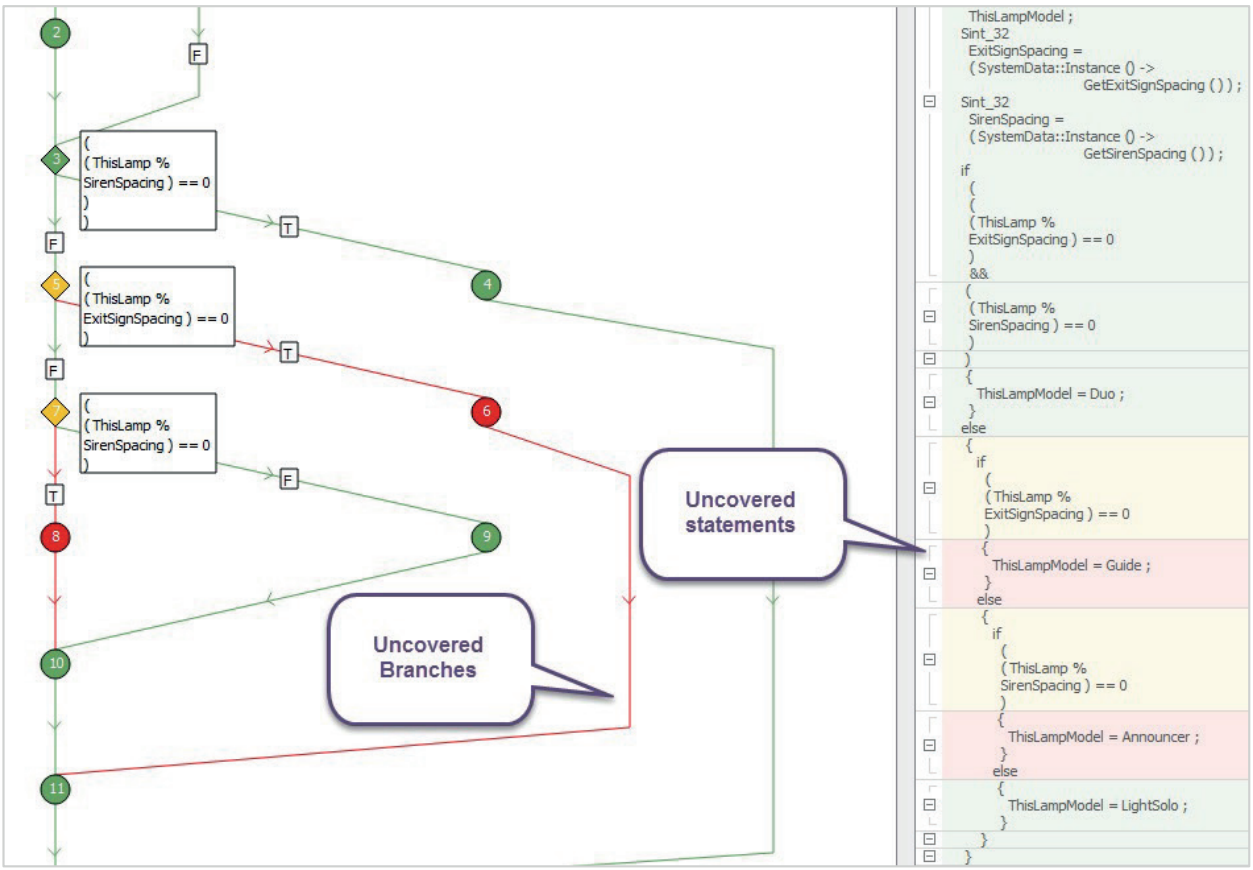


Figure 16: Graphical visualization of code coverage in a flow graph in the LDRA tool suite

Comprehensive functional test and structural coverage artefacts help support the assertion that system requirements are complete and traceable throughout development.

Structural coverage can take several forms. DO-178C objectives A7-5, 6, and 7 relate to the achievement of 100% MC/DC, decision, and statement coverage respectively. The higher the DAL the more demanding the structural coverage objectives, with DAL A requiring the analysis of object code in addition to source code [35].

Efficient instrumentation and MCPs

If coverage analysis is to be performed at the software level, then probes are required within that code to track the execution. That inevitably has an impact on both performance and resources.

For instrumentation to be effective in a multicore system, that impact must be minimized. LDRA's instrumentation techniques rely on probes inserted into each basic block, meaning that there are as few of them as possible (Figure 17).

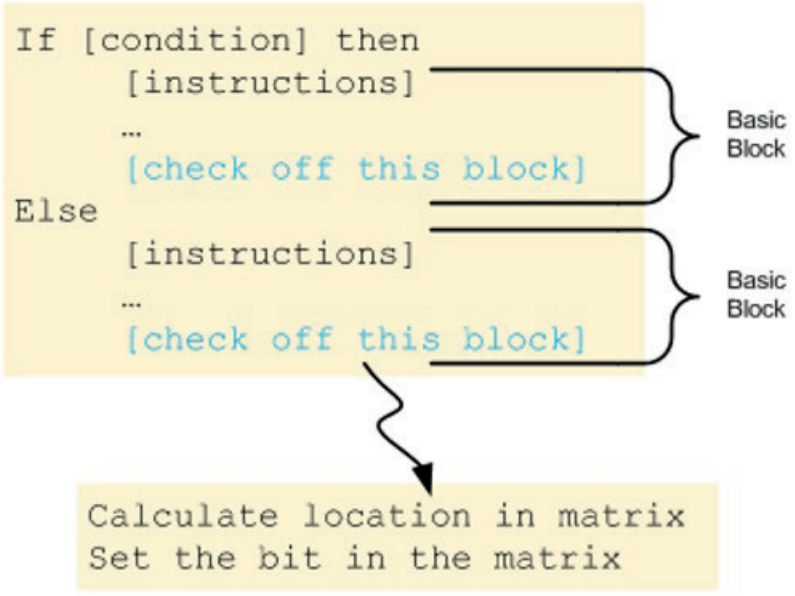


Figure 17: Efficient instrumentation involves inserting one probe per basic block.

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 18).

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

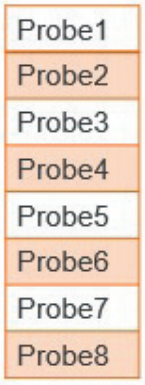


Figure 18: Bit-packed coverage data storage

In a multicore system, collisions are possible as more than one core writes to the same location, when the second attempt will fail. However, because embedded systems generally run in a loop, it is highly likely that any probe that was missed previously will be executed again. If that does not happen, there will always have been at least as much code exercised as has been reported.

The alternative approach, involving semaphores or mutexes, would have a significant performance overhead. The only exception involves MC/DC analysis, where atomic locking is required to ensure 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, especially with respect to hardware interference. The provision of robust partitioning between domains using an RTOS, hypervisor, or separation kernel is likely to be helpful. That said, DO-178C dictates that the responsibility for demonstrating adequate resourcing remains firmly with the application developer.

One recommendation is the use of an iterative development model to overcome this challenge, repeatedly testing and adapting the system as it is developed ensures that the potential for such interference is minimized, and hence that WCET constraints are met. 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 such as the TBmanager component of the LDRA tool suite is invaluable.

In the case of civil aviation projects, adherence to CAST-32A & A(M)C 20-193 will be in tandem with compliance to other standards too, typically DO-178C. In general, the tools required to show adherence to that standard are no different, but some characteristics are critical. For example, care needs to be taken over the efficiency of instrumentation to avoid it being detrimental to the operation of the system as a whole.

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

Works cited

[1] Federal Aviation Administration (FAA), “Certification Authorities Software Team (CAST),” 18 August 2020. [Online]. Available: faa.gov/aircraft/air_cert/design_approvals/air_software/cast/. [Accessed 11 November 2021].

[2] Federal Aviation Administation (FAA), “Federal Aviation Administation (FAA),” [Online]. Available: https://www.faa.gov/. [Accessed 3 November 2021].

[3] European Union Aviation Safety Agency (EASA), “European Union Aviation Safety Agency (EASA),” 2021. [Online]. Available: https://www.easa.europa.eu/. [Accessed 3 November 2021].

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

[5] 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].

[6] F. Wolfe, “EASA and FAA to Issue Further Guidance on Multicore Certification This Year,” 28 February 2020. [Online]. Available: https://www.aviationtoday.com/2020/02/28/easa-and-faa-to-issue-further-guidance-on-multicore-certification-this-year/. [Accessed 4 November 2021].

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

[8] 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].

[9] 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].

[10] QNX, “Multicore Processing,” [Online]. Available: http://www.qnx.com/developers/docs/qnxcar2/topic/com.qnx.doc.neutrino.sys_arch/topic/smp.html. [Accessed 11 November 2021].

[[11] D. Firesmith, “Carnegie Mellon University SEI Blog - Multicore Processing,” 21 August 2017. [Online]. Available: https://insights.sei.cmu.edu/blog/multicore-processing/. [Accessed 11 November 2021].

[12] 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].

[13] 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.

[14] T. Lovelace, “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].

[15] AMD Xilinx, “Zynq™ UltraScale+™ MPSoC,” 2023. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html. [Accessed 23rd February 2023].

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

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

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

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

[20] 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].

[21] DDC-I, “Deos, a Time & Space Partitioned, Multi-core Enabled, RTOS Verified to DO-178C/ED-12C DAL A,” 2021. [Online]. Available https://www.ddci.com/products_deos_do_178c_arinc_653/. [Accessed 4 November 2021].

[22] Green Hills Software, “INTEGRITY-178 tuMP RTOS,” 1996-2021. [Online]. Available: https://www.ghs.com/products/safety_critical/integrity_178_tump.html. [Accessed 4 November 2021].

[23] Lynx Software Technologies, “LynxOS-178,” 2020. [Online]. Available: https://www.lynx.com/products/lynxos-178-do-178c-certified-posix-rtos. [Accessed 4 November 2021].

[24] Lynx, “Lynx MOSA.ic for avionics,” 2020. [Online]. Available: https://www.lynx.com/products/lynx-mosaic-for-avionics-systems. [Accessed 4 November 2021].

[25] Sysgo, “PikeOS RTOS & Hypervisor,” [Online]. Available: https://www.sysgo.com/pikeos. [Accessed 4 November 2021].

[26] RTCA, “DO-297 - Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations,” 8 November 2005. [Online]. Available: https://my.rtca.org/NC__Product?id=a1B36000001IchGEAS. [Accessed 6 January 2022].

[27] Steven H. VanderLeest; Jesse Millwood; Christopher Guikema, “A Framework for Analyzing Shared Resource Interference in a Multicore System,” in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, London, UK, 2018.

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

[29] “AeroSpace Testing Interview Questions,” 27 March 2014. [Online]. Available: http://aerointerview.com/2014/03/testing-interview-questions/. [Accessed 2021 November 8].

[30] MISRA, “MISRA C:2012 Third Edition, First Revision,” The MISRA Consortium Limited, February 2019. [Online]. Available: https://www.misra.org.uk/product/misra-c2012-third-edition-first-revision/. [Accessed 24 September 2021].

[[31] MISRA, “MISRA C++:2008,” The MISRA Consortium Limited, June 2008. [Online]. Available: https://www.misra.org.uk/product/misra-c2008/. [Accessed 24 September 2021].
[32] Lockheed Martin Corporation, JOINT STRIKE FIGHTER AIR VEHICLE C++ CODING STANDARDS FOR THE SYSTEM DEVELOPMENT AND DEMONSTRATION PROGRAM, Lockheed Martin Corporation, 2005.
[33] Carnegie Mellon University Software Engineering Institute, “SEI CERT C Coding Standard,” Carnegie Mellon University Software Engineering Institute, 5 December 2018. [Online]. Available: https://wiki.sei.cmu.edu/confluence/display/c. [Accessed 6 January 2022].
[34] Carnegie Mellon university Software Engineering Institute, “SEI CERT C++ Coding Standard,” Carnegie Mellon university Software Engineering Institute, 29 May 2020. [Online]. Available: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682. [Accessed 6 January 2022].
[35] LDRA, “Object Code Verification,” 2023. [Online]. Available: https://ldra.com/capabilities/object-code-verification/. [Accessed 8th March 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

CAST-32A & A(M)C 20-193 s White Paper v2.1 06/23