



LDRA



**WCET and Critical Applications:
Learning from civil aviation**

mark.richardson@ldra.com



**SOFTWARE PRODUCT ASSURANCE
WORKSHOP 2023**

LDRA Introduction

Provider of Software Quality, Compliance Management & Testing Solutions

Static Analysis for safety and security coding standards compliance
Best-in-class MC/DC coverage
Source code to object code coverage

Automate compliance with standards

ECSS-E-ST-40C & EN 16602-80
ECSS-Q-ST-80C Rev 1 & EN 16602-80
NASA-NPR-7150.2D
NASA-STD-8739.8B

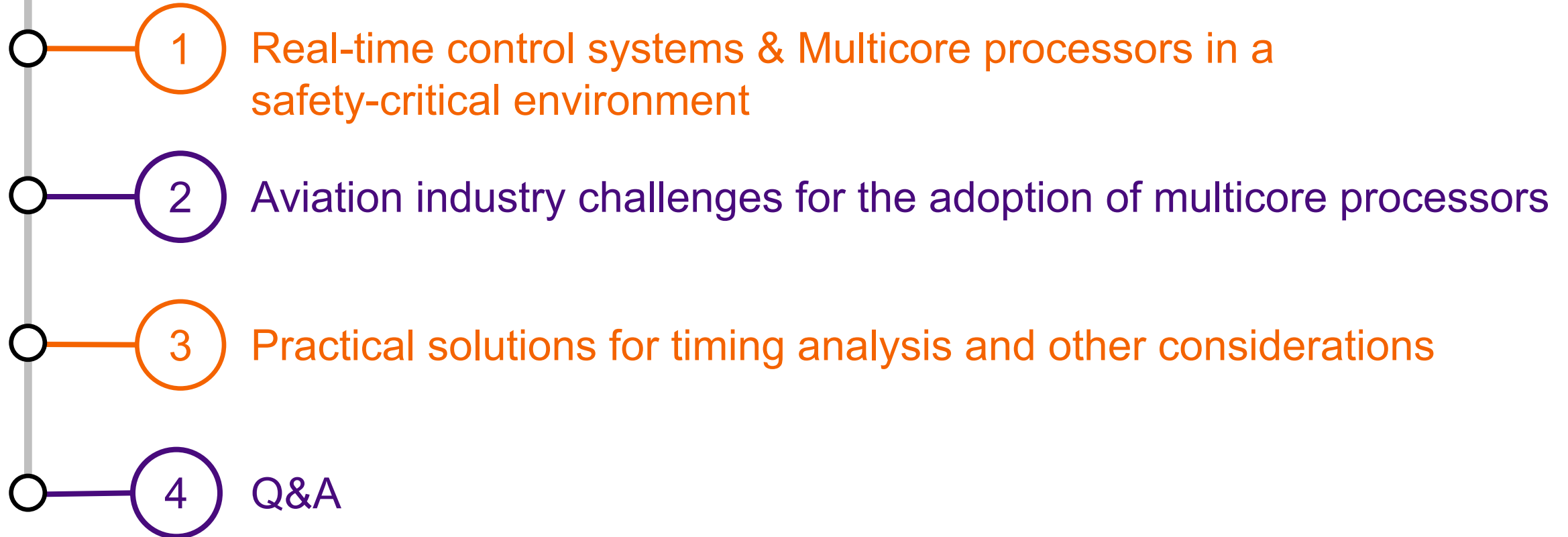
Tried and tested

ESA EGNOS
Euclid Mission
NASA Orion/Artemis
China's Shenzhou VI
ISRO Human Spaceflight Program

LDRA



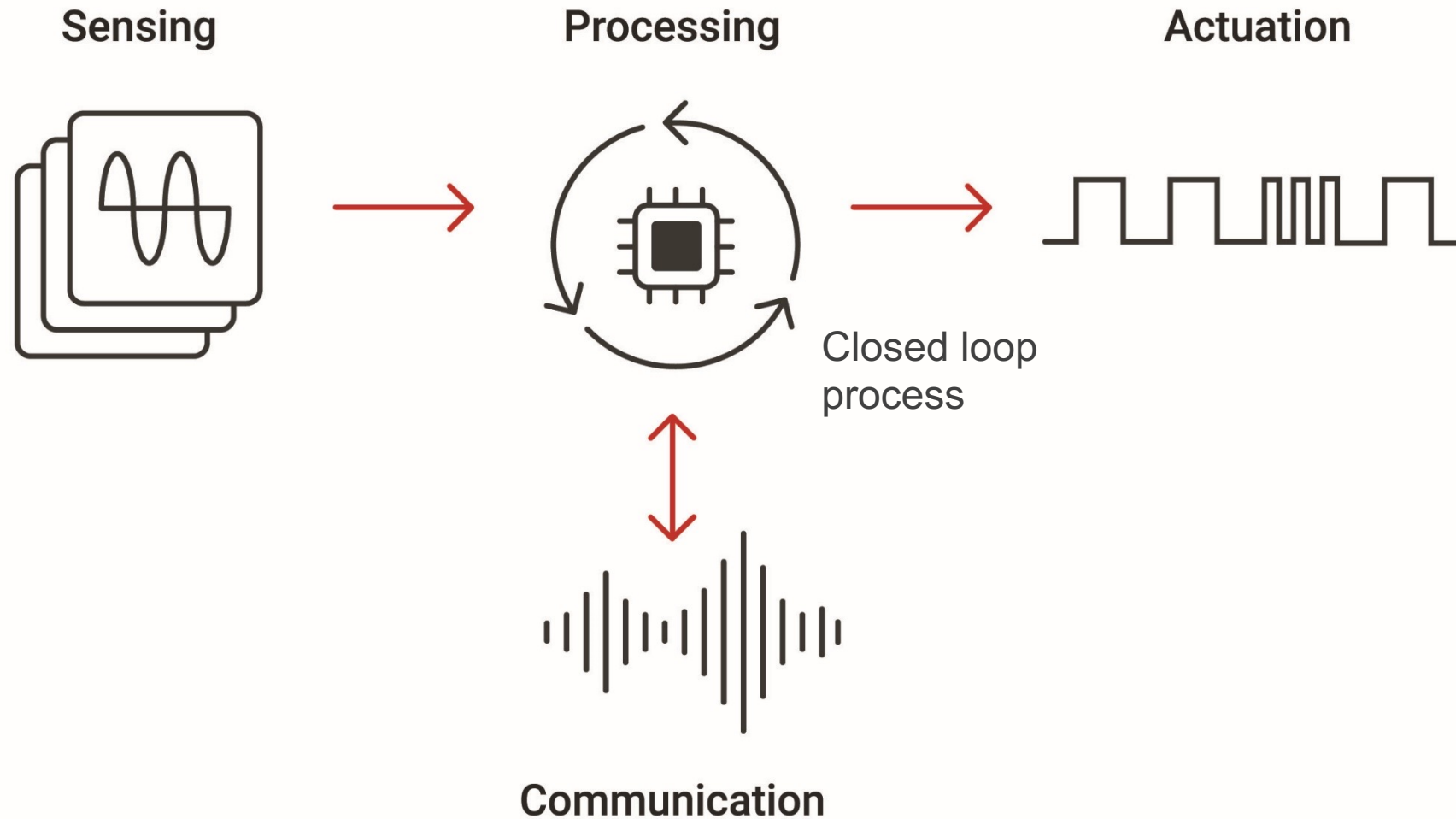
Experts in Safety and Security
Critical Software

- 
- 1 Real-time control systems & Multicore processors in a safety-critical environment
 - 2 Aviation industry challenges for the adoption of multicore processors
 - 3 Practical solutions for timing analysis and other considerations
 - 4 Q&A

**Real-time
control
systems &
multicore
processors in a
safety critical
environment**



[This Photo](#) by Unknown Author is licensed under [CC BY](#)



The closed loop process must complete its execution within an allotted time. Failure to do so will degrade system performance.

Hard-real time versus soft-real time



<https://www>

<https://www.united.com/ual/en/us/fly/travel/inflight/entertainment.html>

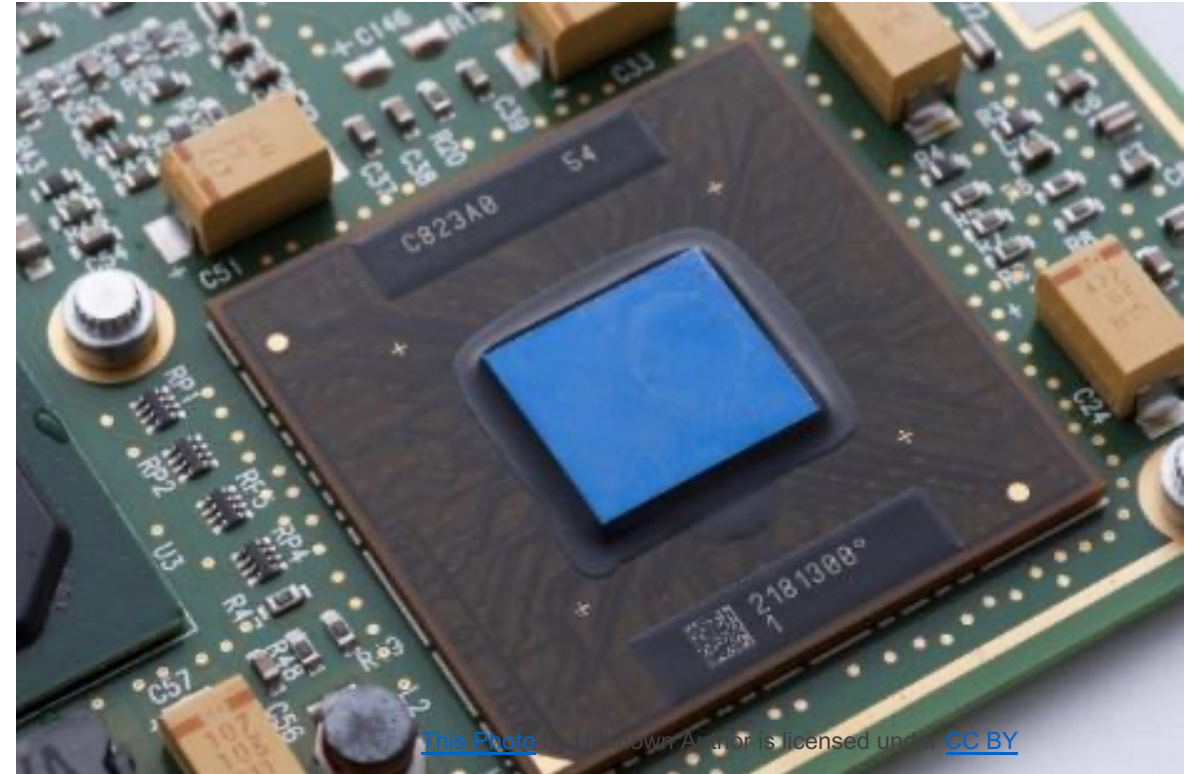
BASIS FOR COMPARISON	HARD REAL TIME SYSTEM	SOFT REAL TIME SYSTEM
Basic	Employs extreme stringent requirements.	Less restrictive
Outcome of the deadline miss	Disastrous if system fails.	The system failure does not result in severe harm.
Usefulness	Reduces abruptly with the increase in tardiness.	Decreases gradually as tardiness increases.
Failure to hit the deadline	Does never occur in hard real-time system (Deterministic).	At times (Probabilistic).
Timing constraints	Hard when user requires validation.	Soft if there is no requirement of validation.
Quality of service provided	Temporal	Best

<https://techdifferences.com/difference-between-hard-and-soft-real-time-systems.html>



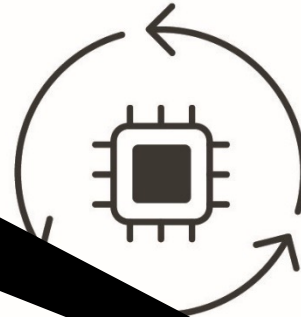
**Too many cooks
spoil the broth**

- Conventionally, single-core processors have been utilized for decades in the context of hard real-time aviation systems
- But multicore processors (MCPs) offer several significant benefits:
 - Performance
 - Power consumption
 - Form-factor & size
 - Availability
- However, the use of a multicore processor introduces the risk of possible interference between processing cores, due to shared resources



Execution time is shared so that these tasks each has short bursts of the execution time on a single core

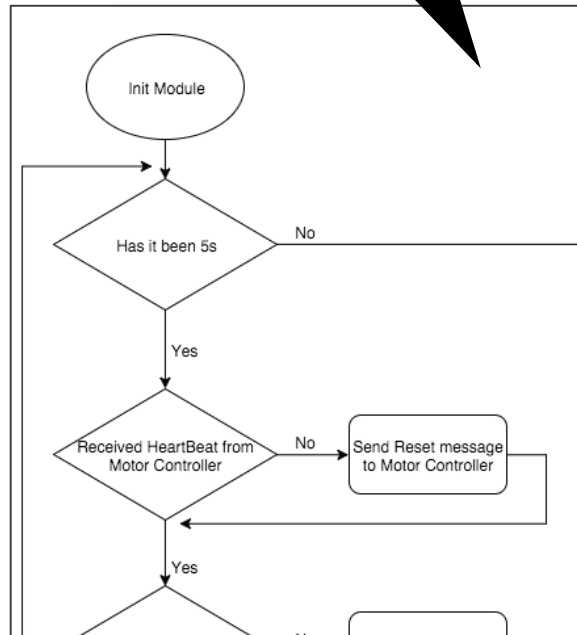
Processing



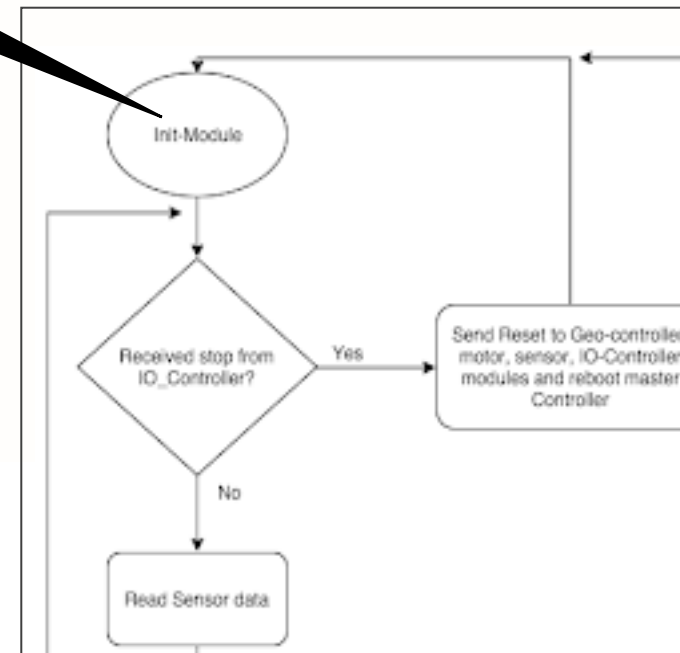
Actuation



1 Hz task flowchart



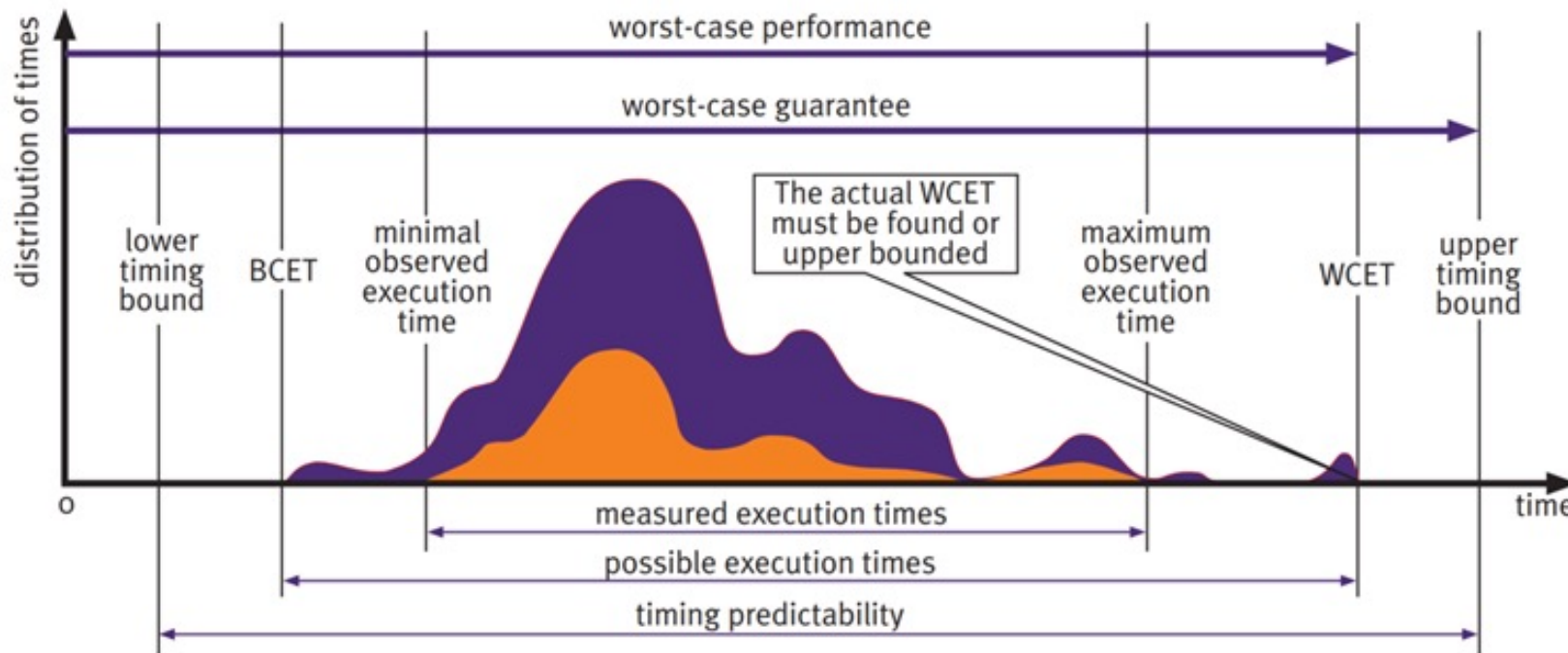
10 Hz task flowchart



Communication



- Single-core processors have been utilized for decades in the context of functional safety
- In this environment it is possible* to calculate a “worst case” approximation to allow sufficient CPU headroom (capacity)

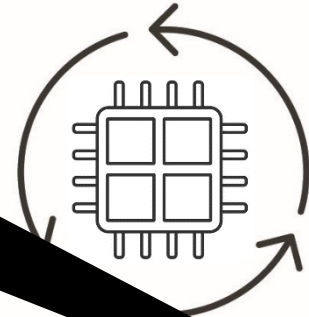


WCET: Worst-Case Execution Time
BCET: Best-Case Execution Time
ACET: Average-Case Execution Time

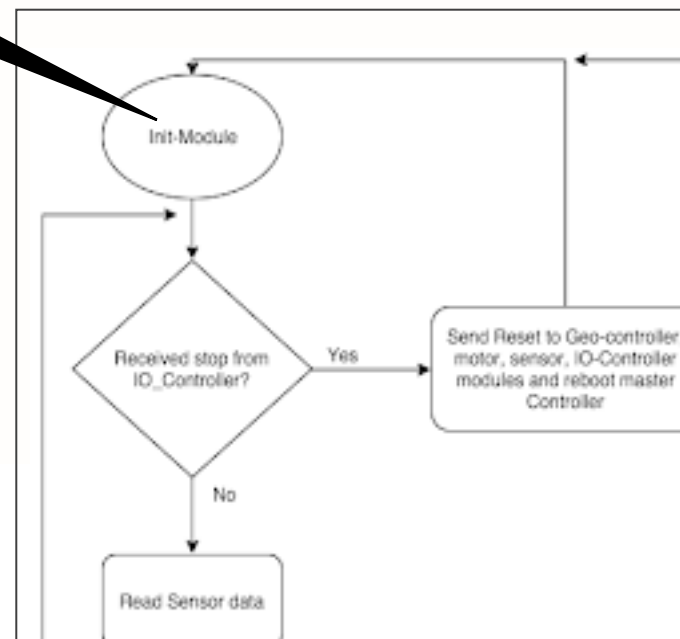
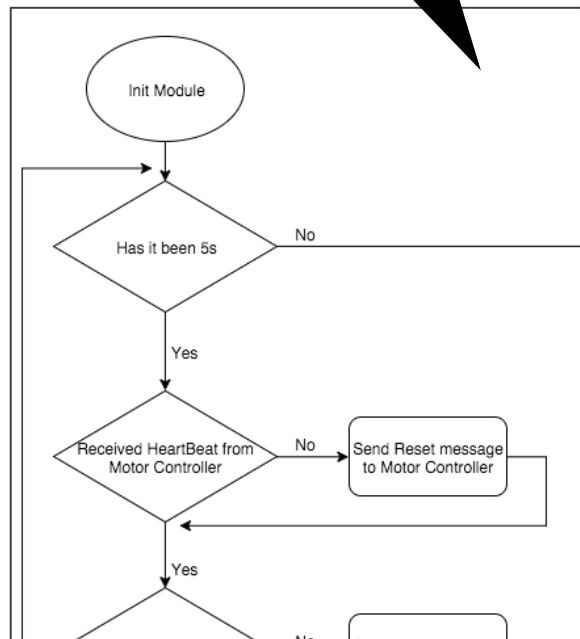
*C. L. LIU and JAMES W. LAYLAND, “Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,” *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46-61, January 1973.

(SIMPLISTICALLY SPEAKING)
Now we can give each of these tasks their own core to run on!

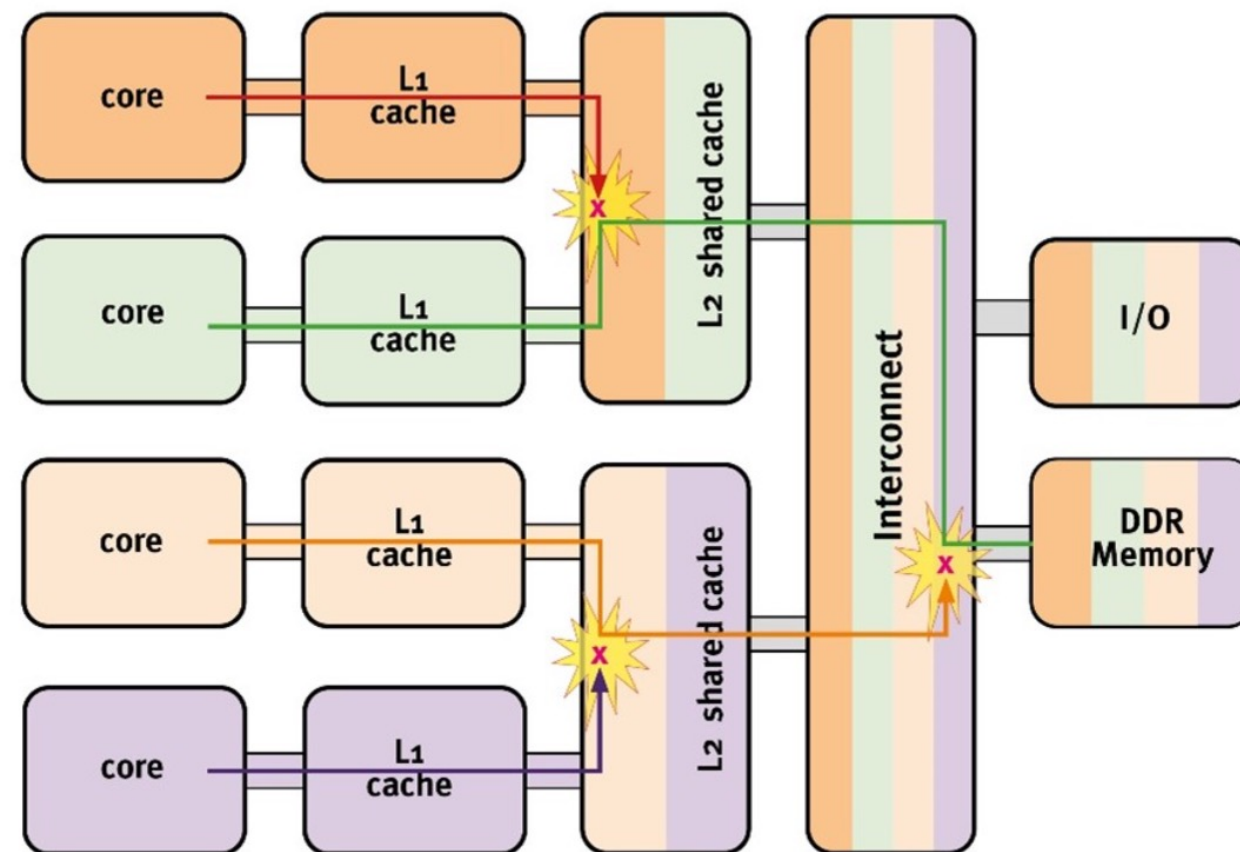
Processing



Actuation

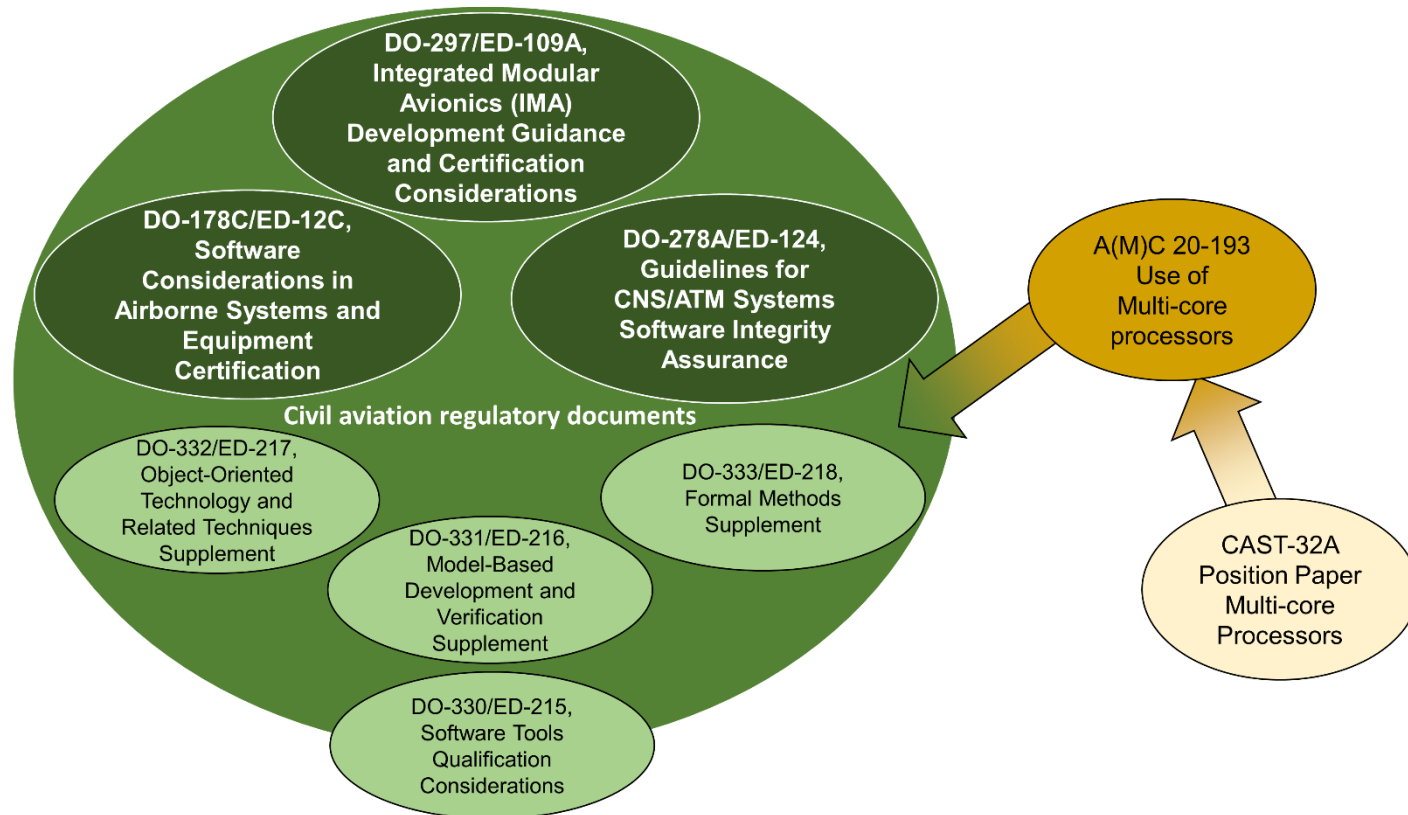


- On a single-core processor, memory is dedicated to that core
- The introduction of additional cores results in those resources being shared between them. Time-related delays occur as users wait for access
- These interference channels cause the execution-time distribution to spread
- Instead of a tight peak, the distribution of execution times becomes wide with a long tail



Aviation industry challenges for the adoption of multicore processors





- Guidance on how to address these issues could initially be found in the CAST-32A position paper
- They have already been formalised into the AMC 20-193 document in Europe (under EASA jurisdiction)
- That formalisation is expected to be mirrored soon in the US (under FAA jurisdiction)

CAST-32A & AMC 20-193 recommendations: WCET considerations

MCP_Resource_Usage_4: The applicant has identified the available MCP and of its interconnect in the intended final configuration, has resources of the MCP to the software applications hosted on the MCP and the demands for the resources of the MCP and of the interconnect do available resources when all the hosted software is executing on the target

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

MCP_Software_1: The applicant has verified that all the software components hosted by the MCP comply with the Applicable Software Guidance. In particular, 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.

The way in which the applicant should demonstrate compliance with this objective depends on the type of the MCP platform:

- **MCP Platforms With Robust Partitioning:**

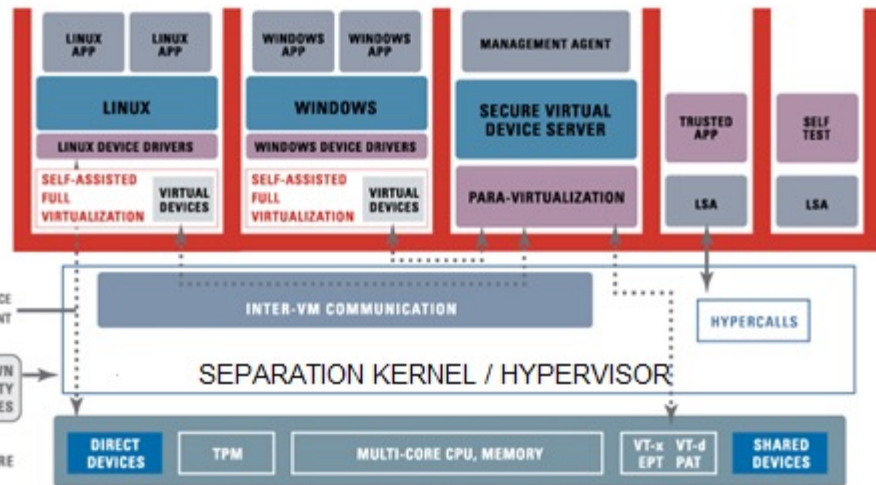
Applicants who have verified that their MCP Platform provides both Robust Resource and Time Partitioning (as defined in this document) may verify applications separately on the MCP and determine their WCETs separately.

- **All Other MCP Platforms:**

Applicants may verify separately on the MCP any software component or set of requirements for which the interference identified in the interference analysis is mitigated or is precluded by design.

Software components or sets of software requirements for which interference is not avoided or mitigated should be tested on the target MCP with all software components executing in the intended final configuration, including robustness testing of the interfaces of the MCP.

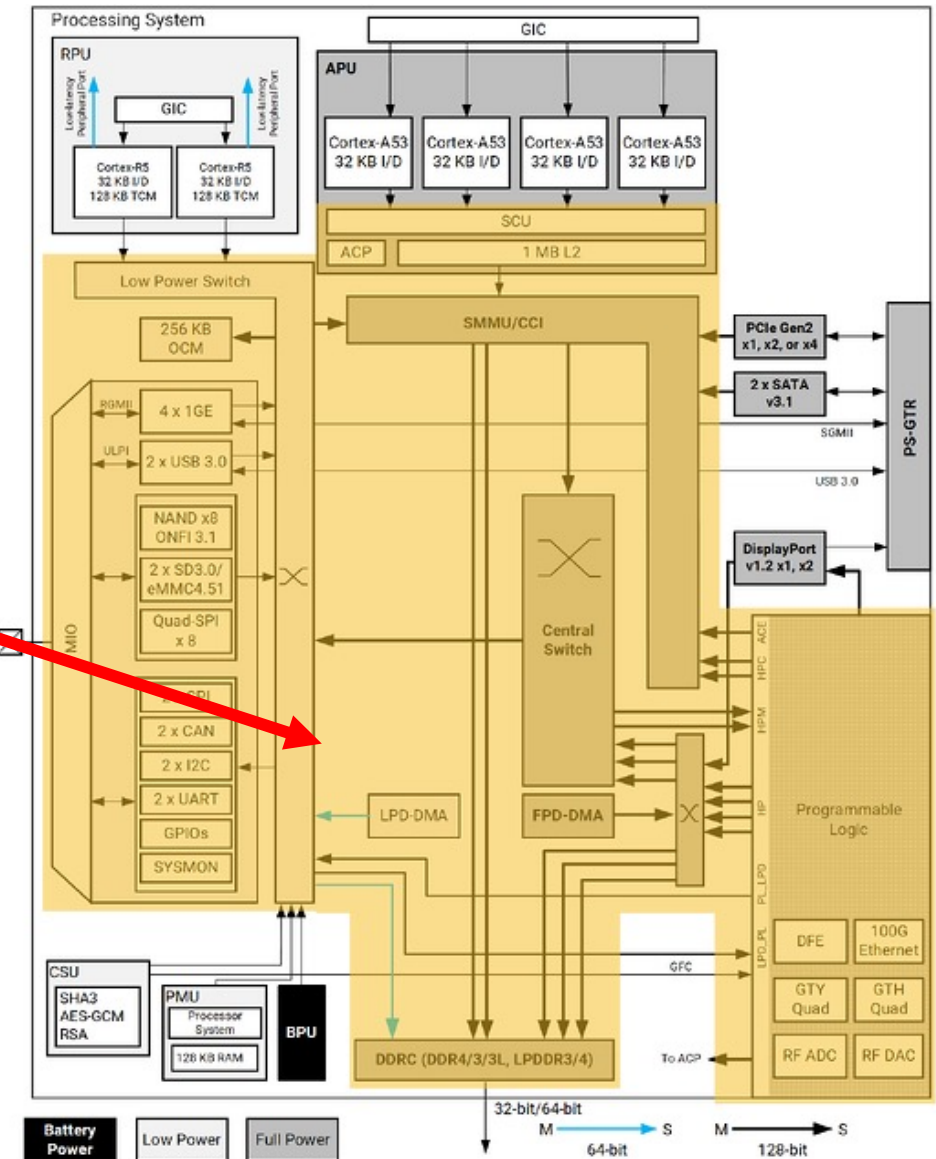
The WCET of a software component may be determined separately on the MCP if the applicant shows that time interference is mitigated for that software component, otherwise, the WCET should be determined by analysis and confirmed by test on the target MCP with all software components executing in the intended final configuration.



But there's more to it than that...

- This is the Xilinx ZYNQ UltraScale+ MPSoC block diagram
- **EVERYTHING** shaded on this diagram is a Hardware Shared Resource (HSR)!
- Relatively few can be explicitly allocated...

[ROBUST PARTITIONING IS DEAD – WHAT NOW? \(lynx.com\)](http://lynx.com)



...and the demands of ECSS-E-ST-40C are quite explicit!

<5.2> Software budget (sizing and timing)

- a. The status of margins regarding the technical budgets shall be presented in the SVR at each milestone, describing the utilized analytical hypothesis.
- b. The margins shall be established by estimation for PDR, by analysis of design after detailed design, and consolidated by performance measurements commensurate with the software implementation for CDR, QR and AR.

Practical solutions for timing analysis and other considerations





Sequence TCI_1_1_2 (C) : Files 1 : Test Cases 1

Log View

The sequence selected was TCI_1_1_2
Assigning stubs
Resolving unassigned stubs
Optimising stubs

File View

TCI_1_1_2

Sequence File Explorer

image_blur.c - /home/ubuntu/ldra_workarea_c_cpp_10.0.3/examples/third_party/wcet/
blur

Calls View

Procedure Calls	Number of Calls	Call Type	() Parameter:	Return Type	{ } Namespace	Class
clock	2	System				
fclose	2	System				
fopen	2	System				
fwrite	1	System				
getc	4	System				
printf	5	System				
putc	3	System				
sizeof	1	System				

Test Case View

Test Case	Regression P / F	Procedure
Tc 1 (100 Repts)		blur

Variable I/O View

Value

- I "ldra.bmp"
- I "ldra_blur.bmp"
- 1
- ldra_qq_duration

"ldra_blur.bmp"

Driver Build & Execution Options

Build Execution General UNIX Argument Macros

Executable Name \$(Exe)

\$(Seqworkdir)\$(SeqName)_\$(ProgFullID)\$(ExeExtension)

Execution Command

run_wcet_remote.sh "\$(Exe)" stress= 1

Start in Directory for the Execution Command \$(Exestartdir)

\$(Sourcedir)

- The wrapper harness allows the code under test to be exercised repeatedly while simultaneously stressing the target device

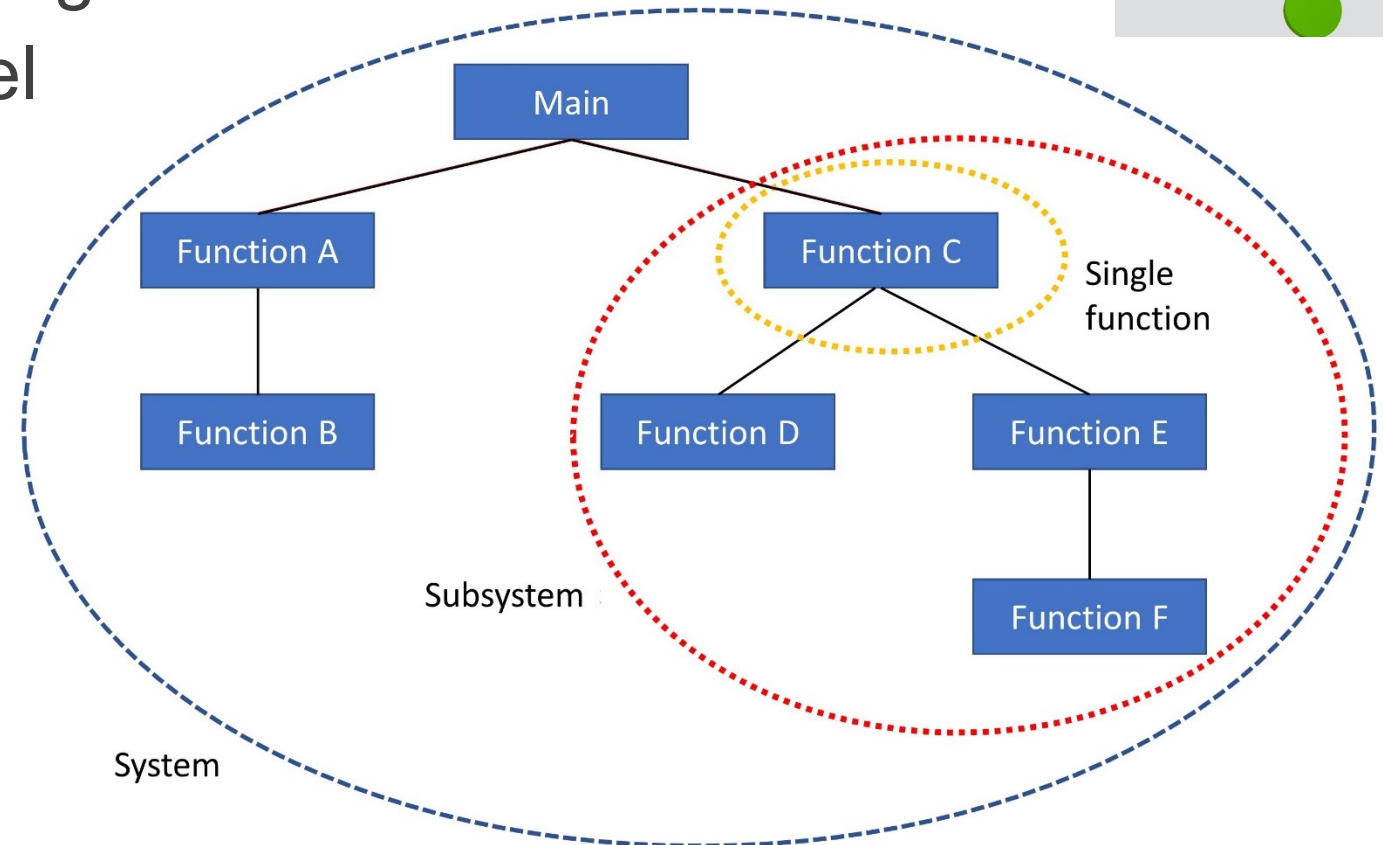
File Name	Description
stress-lockbus.c	core-mwc: add stress_mwc*modn() functions for modulo'd range
stress-lockf.c	stress-lock{a f ofd}: terminate contention process with SIGALRM
stress-lockofd.c	stress-lock{a f ofd}: terminate contention process with SIGALRM
stress-longjmp.c	Update copyright to 2023
stress-loop.c	rework stress_strnrd()
stress-lsearch.c	Update copyright to 2023
stress-madvise.c	core-mwc: add stress_mwc*modn() functions for modulo'd range
stress-malloc.c	core-mwc: add stress_mwc*modn() functions for modulo'd range
stress-matrix-3d.c	stress-matrix-3d: remove redundant redeclaration of variable j
stress-matrix.c	stress-matrix: remove redundant redeclaration of variable j
stress-mcontentd.c	Move common x86 assembler into core-asm-x86.h
stress-membarrier.c	Update copyright to 2023
stress-memcpy.c	rework stress_strnrd()
stress-memfd.c	rework stress_strnrd()
stress-memhotplug.c	stress-*: replace hard coded time constants with #defined constants
stress-memrate.c	stress-memrate: use registers for rep stos, reduces stack loads
stress-memthrash.c	stress-memthrash: add memsetstosd memory zero method
stress-mergesort.c	Update copyright to 2023
stress-mincore.c	stress-*: replace hard coded time constants with #defined constants
stress-misaligned.c	Update copyright to 2023
stress-mknod.c	core-mwc: add stress_mwc*modn() functions for modulo'd range

<https://github.com/ColinlanKing/stress-ng>

- CAST-32A §MCP_Resource_3 suggests identifying HSRs and mapping them to stressors
- A test harness “wrapper” approach includes the flexibility to specify a preferred stressor mechanism – perhaps using [Stress-ng](#)
- There are specific stressors for components such as
 - CPU core operations,
 - built-in registers,
 - cache,
 - ram,
 - virtual memory...

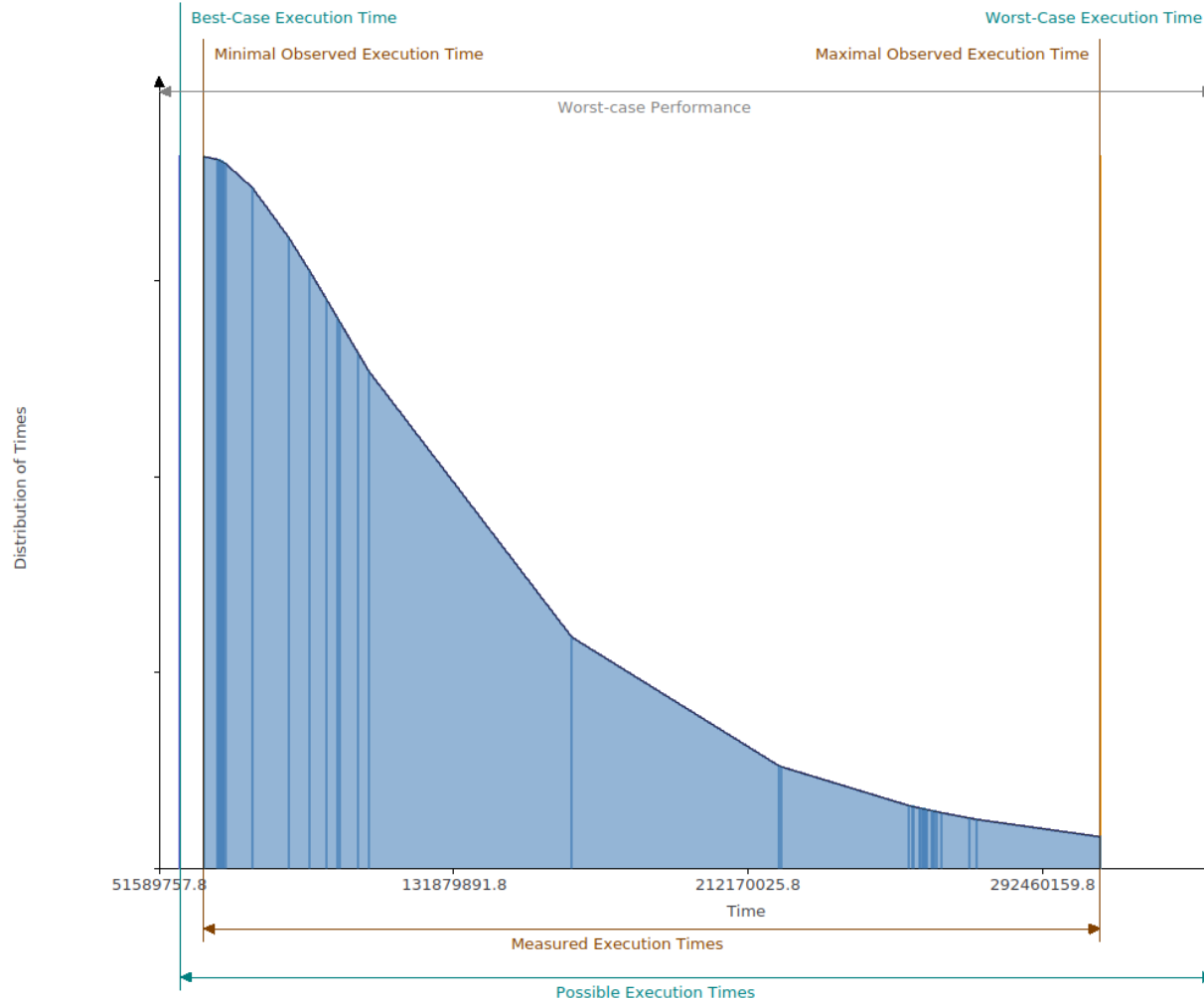


- This “wrapper” principle affords the flexibility to perform execution time analysis from complete system behaviour, through a thread or process, right down to class/function/procedure level
- This approach provides the ability to “drill in” to problem areas, and not just analyse the system as a whole





WCET Diagram : cycle_60hz | Test Case 1

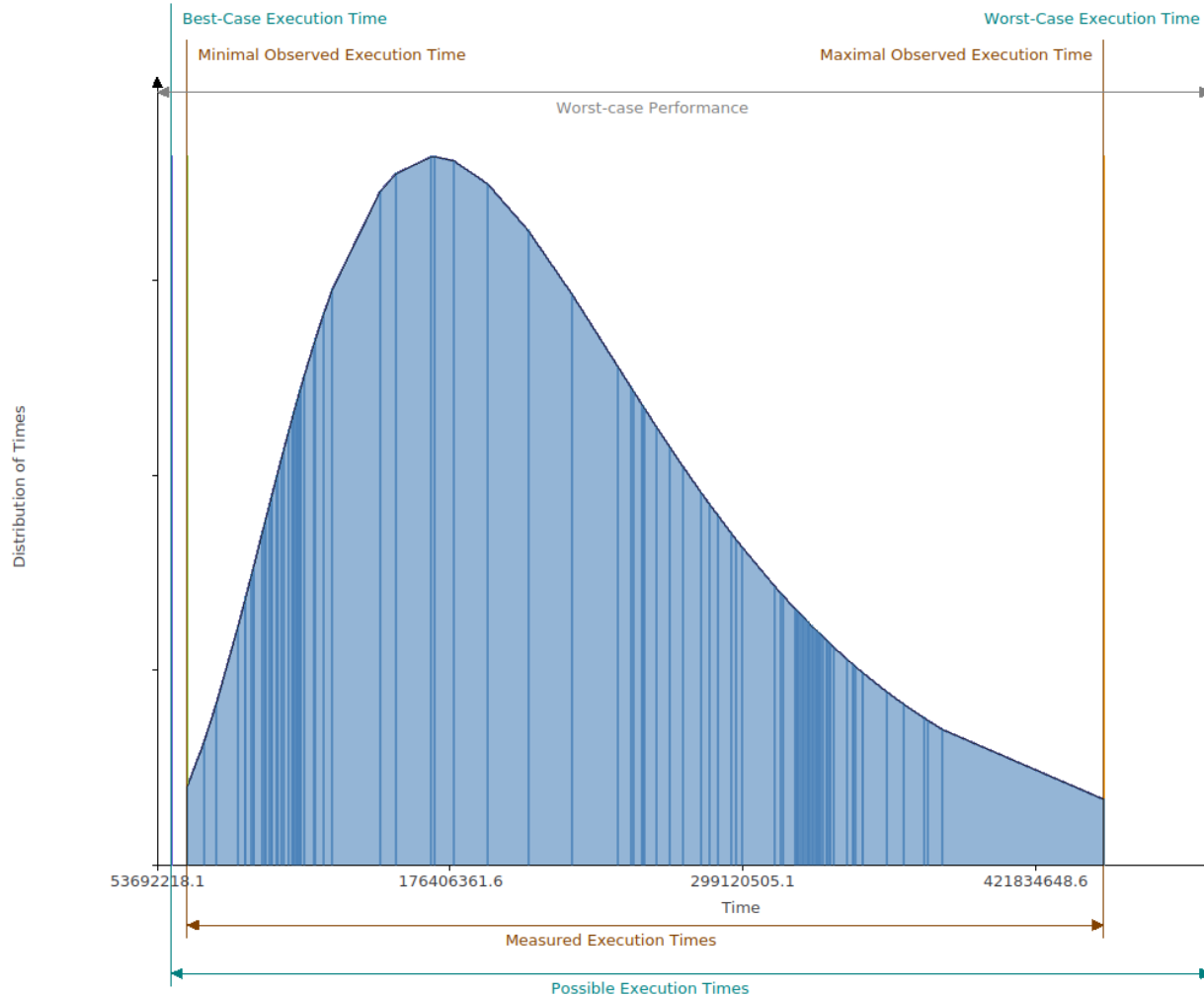


Timing Summary

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



WCET Diagram : cycle_60hz | Test Case 1



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

- Measuring execution times addresses the primary concern
- But that is only one piece of the puzzle
- Dealing with this issue has implications throughout the development lifecycle
- Automated tools can help address those implications, too

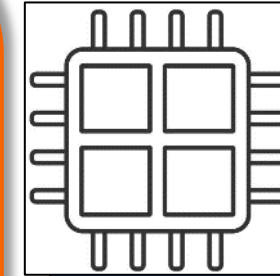


- Execution WCET
- Static analysis provides focus for execution time analysis
- Control and data coupling analysis helps to minimize timing variation
- Requirements traceability keeps tabs on the resulting iterative process



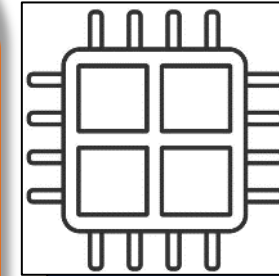


The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art





The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art





The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art



Radiation hardening has resulted in slower MCP adoption in space





The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art



Radiation hardening has resulted in slower MCP adoption in space





The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art



Radiation hardening has resulted in slower MCP adoption in space



That provides an opportunity to learn from CAST-32A & A(M)C 20-193

AMC 20-193 Use of multi-core processors

1. Purpose

1.1 This AMC describes an acceptable means, but not the only means, for showing compliance with the applicable airworthiness specifications for aspects related to multi-core processors (MCPs) contained in airborne systems and equipment used in product certification or ETSO authorisation. Compliance with this AMC is not mandatory, and an applicant may elect to use an alternative means of compliance. However, the alternative means of compliance must meet the relevant requirements, ensure an equivalent level of safety, and be approved by the Agency on a product or ETSO article basis.

1.2 This AMC provides objectives for the demonstration of compliance with the applicable airworthiness specifications for airborne systems and equipment that contain MCPs, according to the applicability in Section 2 of this AMC.

2. Applicability

2.1. This AMC may be used by applicants, design approval holders, and developers of airborne systems and equipment, which contain MCPs, to be installed on type-certified aircraft, engines, and propellers. This also includes developers of ETSO articles.

This AMC applies to systems and equipment that contain MCPs with two or more activated cores for which the item development assurance level (IDAL) of at least one of the software applications hosted by the MCP or of the hardware item that contains the MCP is A, B, or C. The deactivation of cores is handled through the applicable airborne electronic hardware (AEH) guidance.

This AMC does not apply when the IDALs are all Level D or E.

If an applicant modifies the use of the MCP (such as by activating one or more additional cores or



The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art



Radiation hardening has resulted in slower MCP adoption in space



That provides an opportunity to learn from CAST-32A & A(M)C 20-193

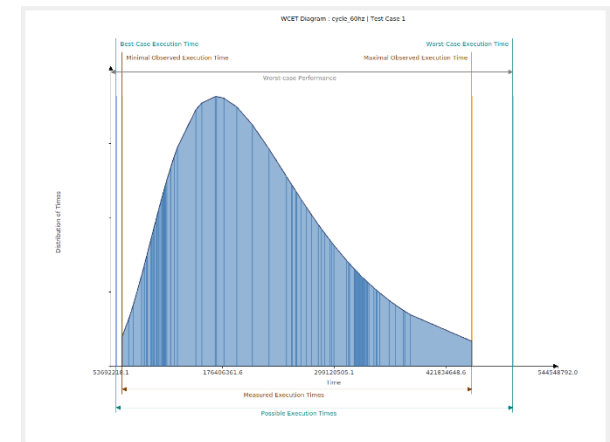


Limiting hardware interference is beneficial – and measuring performance is essential



“The GR765 architecture... contains design extensions to provide hardware support for isolation between mixed-criticality applications.”

<https://www.gaisler.com/index.php/products/components/gr765>



<https://ldra.com/capabilities/mcp/>



The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art



Radiation hardening has resulted in slower MCP adoption in space



That provides an opportunity to learn from CAST-32A & A(M)C 20-193

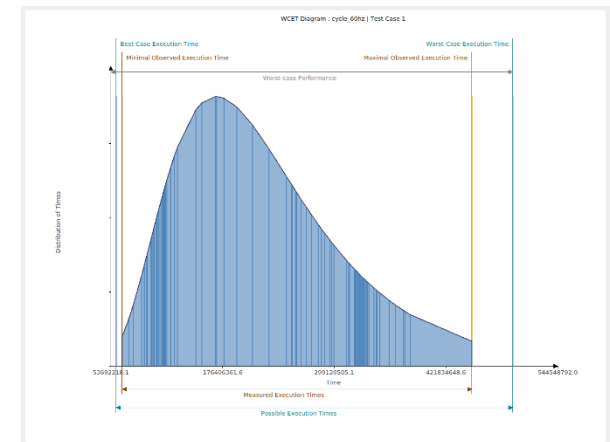


Limiting hardware interference is beneficial – and measuring performance is essential



“The GR765 architecture... contains design extensions to provide hardware support for isolation between mixed-criticality applications.”

<https://www.gaisler.com/index.php/products/components/gr765>



<https://ldra.com/capabilities/mcp/>



The MCP interference problem is not sector-specific – and civil aviation has advanced the state of the art



Radiation hardening has resulted in slower MCP adoption in space



That provides an opportunity to learn from CAST-32A & A(M)C 20-193



Limiting hardware interference is beneficial – and measuring performance is essential



The resulting iterative process impacts the whole development lifecycle





Need more information?



LDRA Software Technology



LDRA Limited



@ldra_technology



LDRA Tools

