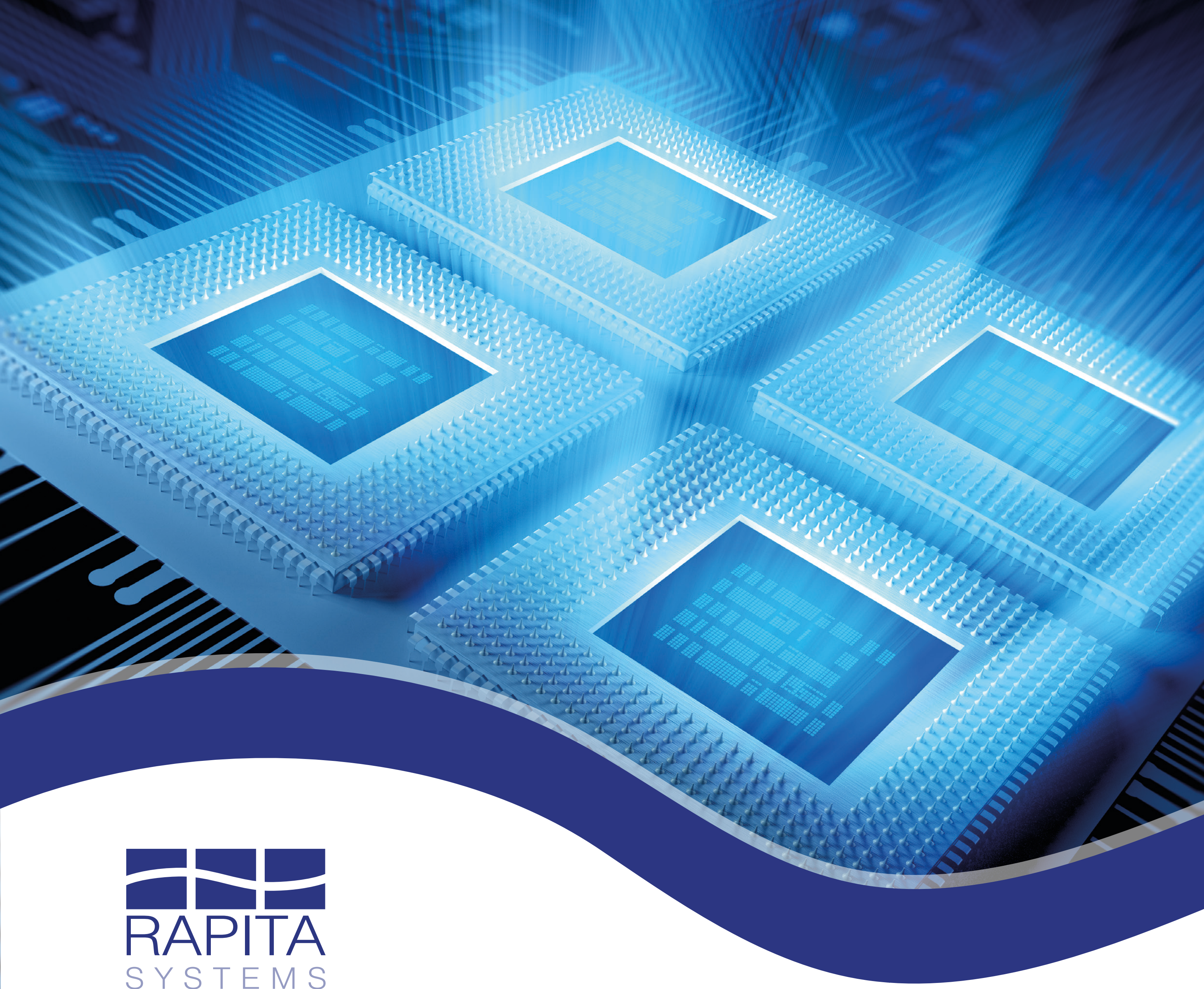


Safety through quality

WHITE PAPER

Multicore Timing Analysis for DO-178C

Document ID: MC-WP-009 MTA for DO-178C v6 INDT-v10

Contents

1. The multicore revolution and DO-178C

Many OEMs are concerned about the long-term component availability of single core processors. This has led some to adopt multicore processors but disable all but one core, as they can't economically verify the system when all cores are enabled.

This isn't a good long-term solution and doesn't take advantage of the performance improvements offered by using multicore hardware to its full potential. The challenges of using multicore processors in the critical embedded domain should be tackled head on, and the potential of these processors embraced.

Since its inception in the 1980's, the guidance offered by DO-178 and its successors has served the avionics industry well. DO-178B, published in 1992, and more recently DO-178C (2011), have kept pace with changes in avionics hardware by ensuring that their guidelines remain generic and relevant regardless of software architecture, programming language, etc.

Since DO-178 was first published, the embedded computing world has seen many significant changes, for example Moore's Law having driven advances such that the computing power of modern cellphones now exceeds that of the Apollo 11 lunar lander many times over. One of the most significant changes is the innovation and use of multicore processors. With a higher density of silicon, these systems offer increased performance per unit area, which is critical to meet the needs of modern avionics systems. Their use comes at a price, as unlike single core systems, they offer neither a deterministic environment nor predictable software execution times.

In response to the increased use of **multicore processors**, the Certification Authorities Software Team (CAST) published Position Paper CAST-32A named 'Multi-core Processors' (often referred to as just 'CAST-32A'). This Position Paper has since been superseded by official guidance – for DO-178C projects certified by the FAA in AC 20-193, and for ED-12C projects certified by EASA in AMC 20-193. AC 20-193, AMC 20-193 and CAST-32A identify topics that could impact the safety, performance and integrity of airborne software systems executing on multicore processors and provide objectives intended to guide the production of safe multicore avionics systems.

For example, and of particular relevance to this paper, objective `MCP_Software_1` in all three guidance documents requires that evidence is produced to demonstrate that all hosted software components function correctly and have sufficient time to complete their execution when operating in their multicore environment. This white paper outlines the challenges in demonstrating this, and presents a practical solution to do so that is compliant with DO-178C, AC 20-193, AMC 20-193 and CAST-32A.

CAST-32A is a Position Paper published by the FAA.

2 key objectives of CAST-32A are relevant to timing analysis:

6.3 Interference Channels and 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."*

6.4 Software Verification

"`MCP_Resource_Usage_4`*: The applicant has identified the available resources of the MCP ... and has verified that the demands for the resources of the MCP and of the interconnect do not exceed the available resources when all the hosted software is executing on the target processor.*

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

CAST-32A in DO178C certification

"The purpose of this CAST paper is to identify topics that could impact the safety, performance and integrity of a software airborne system executing on Multi-Core Processors (MCP)." - FAA

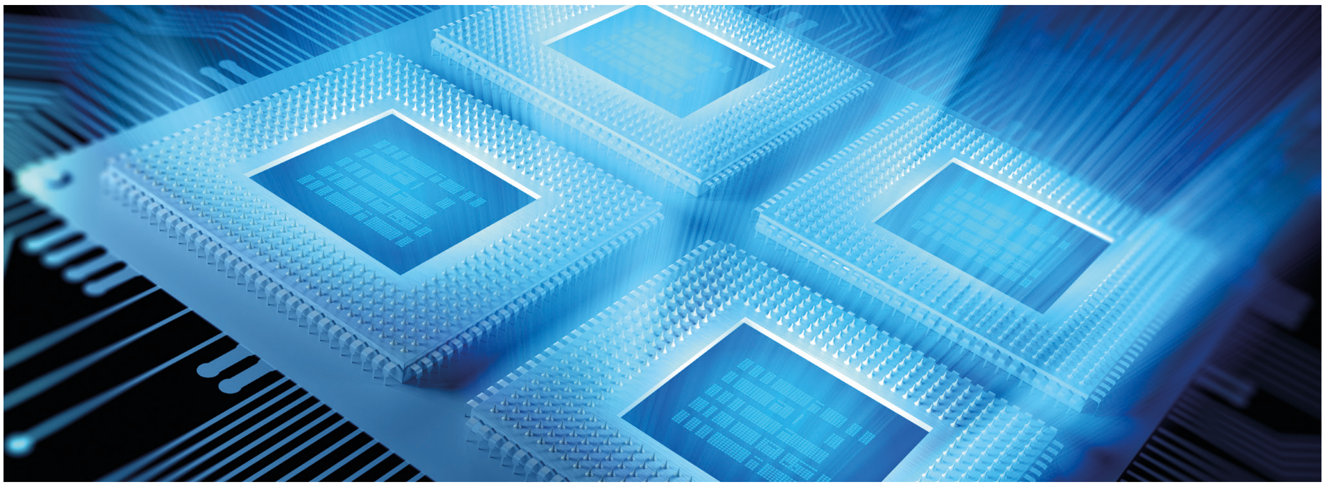
AMC 20-193 vs. CAST-32A

AC 20-193 and AMC 20-193 are largely similar to CAST-32A. The official guidance documents include additional exemptions and clarifications such as how the guidance can be applied for Integrated Modular Avionics (IMA) systems.

You can find out more on our blog.

rapitasystems.com/

2. Challenges of multicore timing analysis



Stay up to date

The Rapita Systems blog addresses topics related to on-target verification including WCET analysis and optimization and structural coverage analysis.

rapitasystems.com/blog

Multicore systems are much more complex than their single core counterparts. To understand how to verify their timing behavior, we must first understand the unique challenges inherent in the analysis.

We've listed some of these below:

- **We need to consider resource contention and interference:** The execution time of a task in a multicore system is affected by contention for shared resources and the interference this causes. To investigate the timing behavior of a multicore system, we need to take this interference into account.
- **Multicore timing analysis can't be entirely automated:** The complexity of multicore processors means that building a fully automated timing analysis solution is unrealistic. While tool support can automate most of the data gathering and analysis processes, engineering wisdom and expertise is needed to understand the system and direct tool usage to produce necessary evidence.
- **We have to test on the real hardware:** Multicore CPUs are complex and often their internals are hidden, making purely analytical models of limited use in understanding their timing behavior. As such, the only way to determine exactly how the processor and its components behave is to measure timing behavior on the system itself.
- **Assumptions need to be tested:** To analyze the timing behavior of a multicore system, you will need to make some assumptions about things such as the interference channels in the system and their effects. After running tests based on these assumptions, you will likely need to reassess those assumptions and rerun tests.

We'll explore these challenges in detail in the next sections.

2.1 We need to consider resource contention and interference

The timing behavior of a task in a multicore system is affected not only by the software running on it and its inputs, but also by contention over resources such as buses, caches and GPUs that are shared with tasks running on other cores. This contention causes interference to the timing behavior of the task.

To demonstrate this, we will use the **YOLO**[1] real-time object detection software on an NVIDIA® Jetson® AGX board that has 8 NVIDIA Carmel cores. YOLO is an open source image recognition application that uses a neural network to identify and classify objects. The neural network calculations are performed on the GPU on the target board. To do this, the frame and neural network information are first loaded into memory and then pushed to the GPU for processing.

We measured the reduction in YOLO's frame rate when running YOLO on one of the cores while we applied sustained accesses on the L3 cache from tasks running on between 1 and 7 contending cores. The minimum frame rate from this series of experiments was almost a factor of 10 slower than when no contention was present, highlighting the importance of considering contention when analyzing the timing behavior of multicore systems.

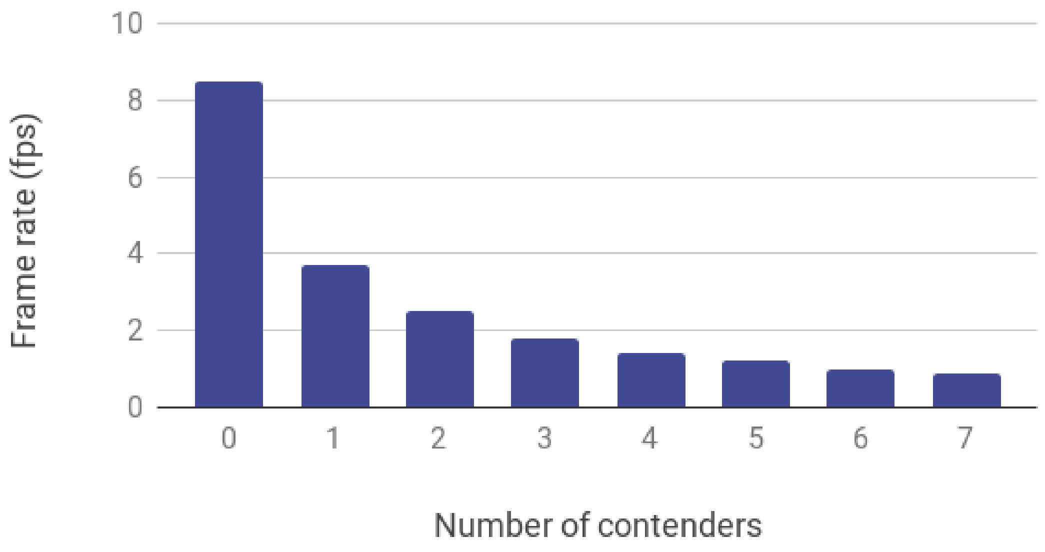


Figure 1 – Frame rate of YOLO with L3 cache contention

[1]https://github.com/pjreddie/darknet

2.2 Multicore timing analysis can't beentirely automated

Timing analysis of single core systems can be entirely automated by using software tools such as Rapi**Time**, which analyze the worst-case execution time (WCET) of tasks running on the system.

This isn't the case for multicore systems, for which we must consider the effects of interference caused by resource contention on software execution times (see *We need to consider resource contention and interference* on page 4). Interference effects are complex, interlinked, and involve components specific to both the multicore architecture and the scheduling and resource allocation systems in the software.

This means that, to properly perform the analysis, we need to apply the expertise of engineers who know the system in detail. While this expertise can be used to direct the use of software tools (for example specifying levels of contention to apply to specific resources), no automated timing analysis tool will be able to understand a multicore system in enough depth to perform the analysis alone.

2.3 We have to test on the real hardware

A measurement-based approach is necessary to obtain execution time evidence for multicore software. Static execution time analysis approaches are not suitable as they require highly detailed models of the processor that are very difficult to obtain and their use would determine the pathological worst-case behavior of the code, which is extremely unlikely to occur.

A measurement-based analysis approach, however, does not rely on models, but instead exercises tests on the multicore hardware itself. Using such an approach, it is possible to collect timing data that reflects the behavior of the system and isn't overly pessimistic.

2.4 Assumptions need to be tested

When beginning to analyze the timing behavior of a multicore system, you'll need to make assumptions about the system, such as interference channels present in it and how they affect each application.

Throughout the analysis process, many of these assumptions may turn out to be invalid, and you'll likely need to use knowledge gained from running tests to feed into a new testing cycle until you can verify that your assumptions are valid.

This is best explained with a practical example. We studied the sensitivity of a memory-intensive application running on a Xilinx® Zynq® Ultrascale+® ZCU102 target board to different levels of interference. The Application Processing Unit on which the application was running has 4 cores.

It would be a reasonable assumption that the L2 cache is a major interference channel for this application due to prior knowledge of the system. To validate this assumption, we design and run a test where the application is running while sustained accesses are made on the L2 cache from tasks running on between 0 and 3 contender cores.

If the assumption is valid, then the number of both L2 cache misses and CPU cycles taken for the application to execute will increase with each additional contender core.

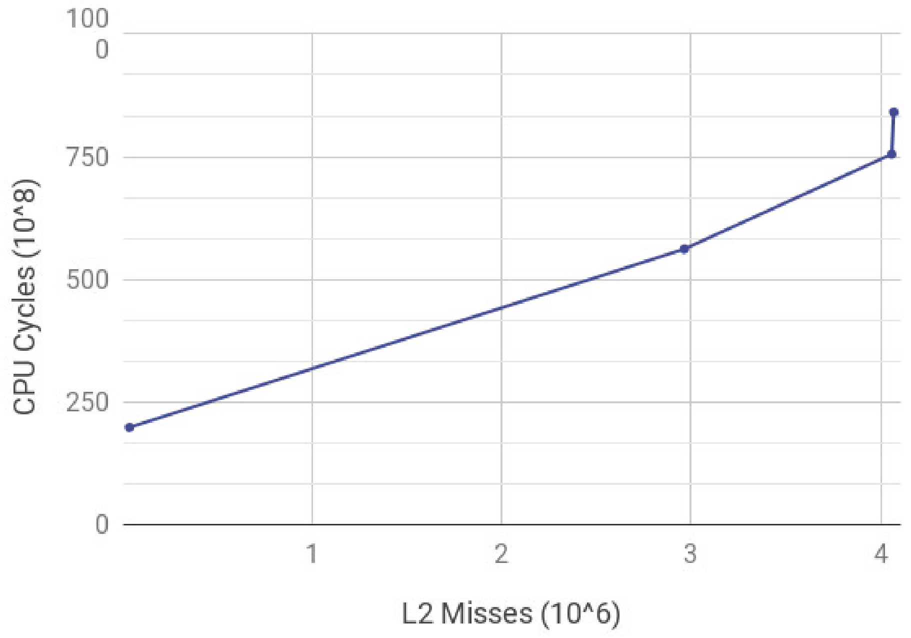


Figure 2 – CPU cycles and L2 cache misses for application on Xilinx Zynq Ultrascale+ ZCU102 board with contention on L2 cache

The figure above shows that the assumption holds until we introduce a third contender core. This increases the number of CPU cycles but the number of L2 cache misses remains around the same as when only two contender cores are active.

This highlights that the assumptions made about how the hardware behaves aren't correct and the system will need to be investigated further to identify why the CPU cycles increase without the L2 cache misses increasing. This could be due to an interference channel that we didn't account for in the analysis, such as a shared bus.

3. The solution

Rapita Systems has developed a solution for multicore timing analysis that meets DO-178C, AC 20-193, AMC 20-193 and CAST-32A guidelines. The solution, MACH^{178} (Multicore Avionics Certification for High-integrity DO-178C projects) features a combination of engineering services and automated tool support to enable the investigation and quantification of interference effects in multicore systems. This allows specific questions about these systems to be answered, and DO-178C certification evidence to be produced.

3.1 The process

MACH^{178} follows a V-model process to produce a clearly structured flow of verification artefacts that satisfy DO-178C traceability requirements and meet AC 20-193 and

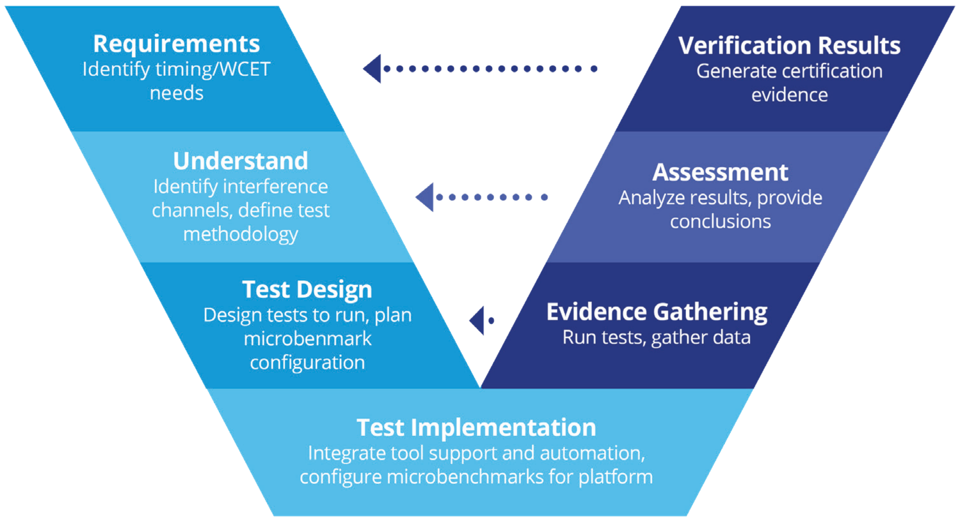

Requirements
Identify timing/WCET needs
Understand
Identify interference channels, define test methodology
Test Design
Design tests to run, plan microbenmark configuration
Test Implementation
Integrate tool support and automation, configure microbenchmarks for platform
Evidence Gathering
Run tests, gather data
Assessment
Analyze results, provide conclusions
Verification Results
Generate certification evidence

Figure 3 – **MACH**^{178} process

AMC 20-193 objectives, ensuring a cost-effective and methodical verification process.

While the stages initially follow a mostly linear progression, many stages may be returned to later in the verification process – for example to redefine assumptions made about the platform under analysis after testing identifies hidden interference channels.

The process is supported by Rapita Systems tooling and services, which are outlined in *Components of the solution* on page 15.

Within the next sections, you'll learn about the process and be able to follow it in context through an example – testing the timing behavior of the Darknet YOLOv3 object recognition application running on an NVIDIA AGX.

3.1.1 Requirements definition

In this stage, we identify high-level verification requirements for the multicore platform and project under analysis.

Requirements depend on the scope and focus of verification activities and what the analysis is to be used for, potentially including:

- Type and level of evidence on timing behavior and other aspects of the multicore system.
- Certification requirements specified by the DO-178C Design Assurance Level (DAL), where appropriate.
- Required verification artefacts depending on certification or other needs.
- Any tool qualification requirements from the multicore analysis.

We use items specified in both AC 20-193, AMC 20-193 and CAST-32A MCP planning phases (**`MCP_Planning_1`** and **`MCP_Planning_2`**) to feed into the requirements elicitation process.

Example: Requirements

Our primary need is to analyze the sensitivity of Darknet YOLOv3's frame rate to interference caused by contention on shared resources. We identify that the minimum frame rate for the application to operate successfully is 1.7 frames per second. From this, we elicit the following requirement:

`REQ-0001` – The frame rate of the Darknet YOLOv3 application shall be no less than 1.7 frames per second when the application must compete for resources.

3.1.2 Understanding the multicore platform

In this stage, we analyze the platform to identify channels of interference that we need to investigate to meet our high-level requirements. We also expand high-level requirements into low-level requirements based on the interference channels we have identified.

Our initial understanding of interference channels present in the system is likely to be incomplete. We use this as a blueprint for building tests in the next stages that reinforce our understanding of the platform under analysis, and may need to revisit this stage later.

We use outputs from this stage to address the AC 20-193/AMC 20-193 MCP resource usage objectives. These objectives require the identification of MCP configuration settings, mitigation mechanisms and interference channels.

Example: Understand

We investigate the NVIDIA AGX hardware and Darknet YOLOv3 software to understand how the system behaves. For demonstration purposes, we provide the simplified analysis below:

- The Darknet YOLOv3 application is an object detection application, which uses a convolutional neural network to perform a significant amount of its calculations using GPU acceleration.
- Performing operations on the GPU typically requires building up a dataset, forwarding it to the GPU for processing and then receiving the results.
- There may be interference from sharing caches and buses.
- The application under analysis does not have to compete for GPU resources with other applications.
- The target board is the NVIDIA Jetson AGX.
- The NVIDIA Jetson AGX has a Xavier SoC.
- The Xavier SoC has 8 Carmel cores (ARMv8 architecture) and a GPU with 512 Tensor cores.

From the information we gathered from our analysis, we expand REQ-0001 to elicit requirements including the following:

- **`REQ-0001-001`**: The frame rate of the Darknet YOLOv3 application shall be no less than 1.7 frames per second when the application must compete for access to the L1 cache.
- **`REQ-0001-002`**: The frame rate of the Darknet YOLOv3 application shall be no less than 1.7 frames per second when the application must compete for access to the L2 cache.
- **`REQ-0001-003`**: The frame rate of the Darknet YOLOv3 application shall be no less than 1.7 frames per second when the application must compete for access to the L3 cache.
- **`REQ-0001-004`**: The frame rate of the Darknet YOLOv3 application shall be no less than 1.7 frames per second when the application must compete for access to buses.

In subsequent stages of the example, we'll focus on addressing **`REQ-0001-003`**, demonstrating how we investigate the impact of contention on the L3 cache on the execution of Darknet YOLOv3.

3.1.3 Test design

In subsequent stages of the example, we'll focus on addressing **`REQ-0001-003`**, demonstrating how we investigate the impact of contention on the L3 cache on the execution of Darknet YOLOv3.

In this stage, we design the test cases needed to address the requirements elicited during the previous stage.

We typically design test cases to do one or more of the following:

- Profile an application by executing it in isolation. This provides valuable information about the characteristics of the applications under test.
- Measure an application's sensitivity to different types and levels of resource contention. We do this by executing the application while Rapi**Daemons** (see *Rapi**Daemons*** on page 16) contend for shared resources. This provides valuable information about how different interference channels affect each application.
- Measure the effectiveness of partitioning mechanisms by causing high loads using Rapi**Daemons**.
- Measure the impact of resource contention on the system when all tasks are executing by adding additional load with Rapi**Daemons**.

We apply an independent review process for all test cases to ensure that they meet the requirements they address, and use problem reporting and management processes to ensure issues are tracked and resolved.

RapiDaemons

Rapi**Daemons** are specialized programs designed to generate contention on specific resources such as buses, caches and GPUs. Each Rapi**Daemon** is designed to either maximize or match a desired level of contention for a specific reource.

Example: Test design

In this stage, we write the following test case to address **`REQ-0001-003`**.

In the test case above, we have identified that there is a dependency for a L3 cache contender. We'll implement this contender in the next stage using a Rapi**Daemon**.

Test Case ID	`TP-L3Cache-01`
Requirements	`REQ-0001-003`
Summary	Identify the frame rate of Darknet YOLOv3 when subject to L3 cache contention
Dependencies	L3 cache contender
Verification Strategy	To verify `REQ-0001-003`: Measure the Darknet YOLOv3's frame rate when run in isolation and when contention is applied on the L3 cache from tasks running on additional cores Success criteria: The minimum frame rate observed is no less than 1.7 frames per second.

3.1.4 Test implementation

In this stage, we integrate software and hardware components needed to perform the analysis into the multicore environment (see *Components of the solution* on page 15). This involves integrating R**VS** tools to enable the automated collection of metrics including execution time metrics and performance counters such as cache misses, and Rapi**Daemons** needed to apply contention on shared hardware resources.

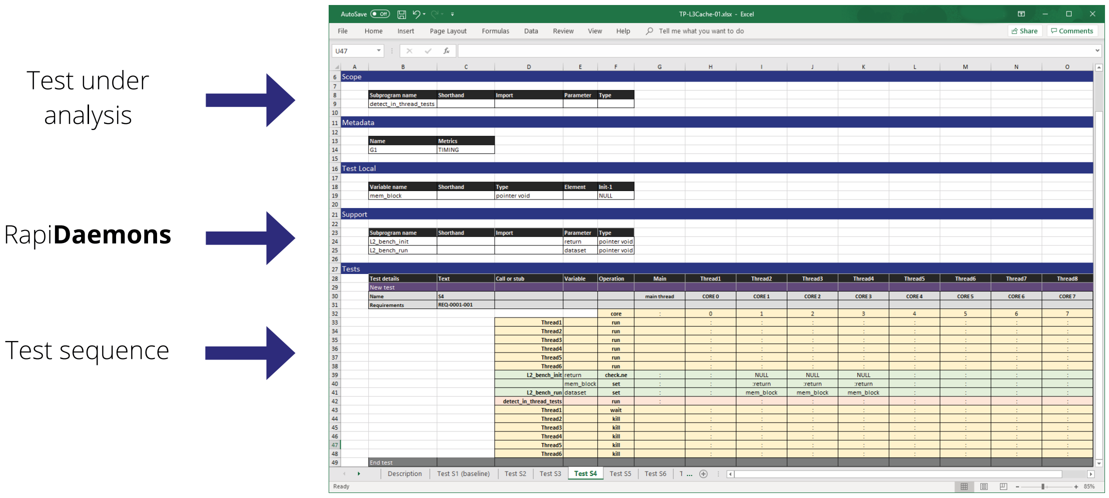
Then, we implement test cases as automated test procedures and execute them. Each test procedure specifies the aspect of the system that we need to measure (e.g. function, task), the set of metrics needed to verify the requirement under test and the specification of when Rapi**Daemons** are to be run, as needed.

We formally test all Rapi**Daemons** on-target to ensure that they are functioning as intended.

Example: Test implementation

We integrate R**VS** with the NVIDIA AGX hardware and Darknet YOLOv3 software and configure a Rapi**Daemon** to apply contention on the L3 cache in this environment. Our integration includes implementing a method to capture other measurements from the system including L3 cache access counters.

We then write test procedures based on our test case, which will collect execution time and L3 cache access information while running the application with contention applied from between 0 and 7 other cores on the NVIDIA Jetson AGX by the use of Rapi**Daemons**.



Test under analysis
RapiDaemons
Test sequence
AutoSave Off
TP-L3Cache-01.xlsx - Excel
File Home Insert Page Layout Formulas Data Review View Help Tell me what you want to do
Share Comments
U47
Description | Test S1 (baseline) | Test S2 | Test S3 | Test S4 | Test S5 | Test S6 | T ...
Ready
85%

Scope

Subprogram name	Shorthand	Import	Parameter	Type
detect_in_thread_tests				

Metadata

Name	Metrics
G1	TIMING

Test Local

Variable name	Shorthand	Type	Element	Init-1
mem_block		pointer void		NULL

Support

Subprogram name	Shorthand	Import	Parameter	Type
L2_bench_init			return	pointer void
L2_bench_run			dataset	pointer void

Tests

Test details	Text	Call or stub	Variable	Operation	Main	Thread1	Thread2	Thread3	Thread4	Thread5	Thread6	Thread7	Thread8
New test													
Name	S4				main thread	CORE 0	CORE 1	CORE 2	CORE 3	CORE 4	CORE 5	CORE 6	CORE 7
Requirements	REQ-0001-001												
				core	:	0	1	2	3	4	5	6	7
		Thread1		run		:	:	:	:	:	:	:	:
		Thread2		run		:	:	:	:	:	:	:	:
		Thread3		run		:	:	:	:	:	:	:	:
		Thread4		run		:	:	:	:	:	:	:	:
		Thread5		run		:	:	:	:	:	:	:	:
		Thread6		run		:	:	:	:	:	:	:	:
		L2_bench_init	return	check.ne	:	:	NULL	NULL	NULL	:	:	:	:
			mem_block	set	:	:	:return	:return	:return	:	:	:	:
		L2_bench_run	dataset	set	:	:	mem_block	mem_block	mem_block	:	:	:	:
		detect_in_thread_tests		run	:		:	:	:	:	:	:	:
		Thread1		wait		:	:	:	:	:	:	:	:
		Thread2		kill		:	:	:	:	:	:	:	:
		Thread3		kill		:	:	:	:	:	:	:	:
		Thread4		kill		:	:	:	:	:	:	:	:
		Thread5		kill		:	:	:	:	:	:	:	:
		Thread6		kill		:	:	:	:	:	:	:	:
End test													

Figure 4 – Test case that collects executing time and L3 cache access information

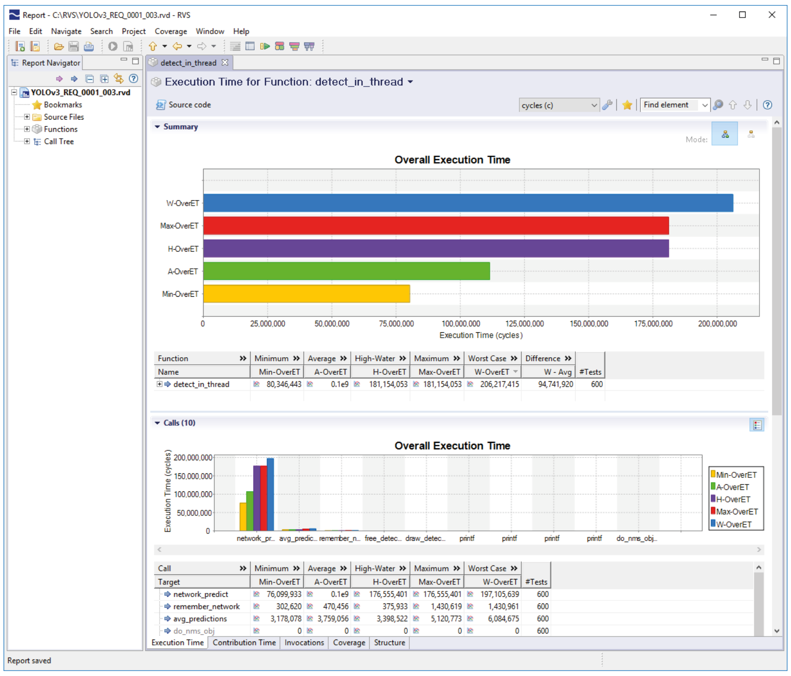
3.1.5 Evidence gathering

In this stage, we run test procedures on-target and collect results. We inspect results to ensure that they are complete and as expected, and reiterate test design where appropriate.

Example: Evidence gathering

We use Rapi**Test** to automatically convert our test procedures into test harnesses that we can run to execute the application under analysis while resource contention is applied, and then run the object files on the NVIDIA AGX.

While Rapi**Test** runs our tests, Rapi**Time** automatically collects an execution trace and processes this to produce an R**VS** report that includes execution time metrics and performance counters that capture additional measurements, including L3 cache access counters.



Function	Minimum	Average	High-Water	Maximum	Worst Case	Difference	
Name	Min-OverET	A-OverET	H-OverET	Max-OverET	W-OverET	W - Avg	#Tests
detect_in_thread	80,346,443	0.1e9	181,154,053	181,154,053	206,217,415	94,741,920	600

Call	Minimum	Average	High-Water	Maximum	Worst Case	
Target	Min-OverET	A-OverET	H-OverET	Max-OverET	W-OverET	#Tests
network_predict	76,099,933	0.1e9	176,555,401	176,555,401	197,105,639	600
remember_network	302,620	470,456	375,933	1,430,619	1,430,961	600
avg_predictions	3,178,078	3,759,056	3,398,522	5,120,773	6,084,675	600
do_nms_obj	0	0	0	0	0	600

3.1.6 Assessment

In this stage, we assess whether the results provide conclusive evidence towards verification of relevant low-level timing requirements.

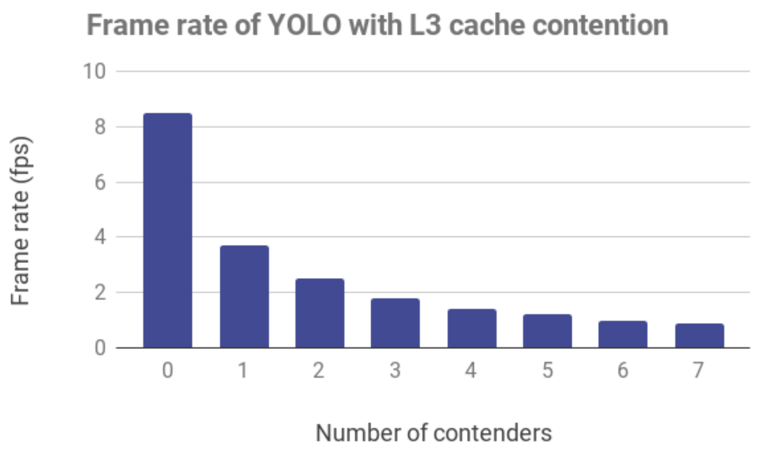
If we encounter any deviations from expected results, we investigate them, revising our understanding of the platform and reformulating our assumptions (see *Understanding the multicore platform* on page 9). For example, assessment of results can identify hidden interference channels in the system, which we will investigate, leading us to design and implement more tests where necessary until we have conclusive evidence that our assumptions about the system are valid.

The outputs from this stage support the AC 20-193/AMC 20-193 **`MCP_Software_1`** objective, which requires that we verify the correct behavior of the hosted software within the allowed time frames.

Example: Assessment

We assess our test results in the context of Requirement **`REQ-0001-003`**, which is being addressed by our test case. The plot below summarizes the results from executing the task under analysis when contention is applied on the L3 cache from tasks running on between 1 and 7 other cores.

From our results, we conclude that the Darknet YOLOv3 application is resilient to contention on the L3 cache when tasks running on up to 3 other cores perform sustained accesses to it. Increasing the number of contenders to 4, however, causes the frame to go below the threshold of 1.7 frames per second specified in **`REQ-0001-003`**. We have also identified that the minimum frame rate from L3 contention with this setup, 0.9 frames per second.



3.1.7 Verification results

In this stage, we produce a report that summarizes verification results collected during the previous stages. We also include any deviations from the process and highlight significant findings from the analysis in this report.

The results in this report let us demonstrate how we met the high-level objectives identified during the first stage (see *Requirements definition* on page 8), while including full traceability per DO-178C and AC 20-193/AMC 20-193.

If the process is being used in a DO-178C certification context, the outcome of this stage is used to supplement the Software Accomplishment Summary referenced in the AC 20-193/AMC 20-193 **`MCP_Accomplishment_Summary_1`** objective.

Example: Verification results

From our analysis, we have obtained enough information to elicit the worst-possible slowdown that the Darknet YOLOv3 application may experience due to contention on the L3 cache and address requirement REQ-0001-003.

To document our findings, we produce a Software Accomplishment Summary report, which is required by AMC 20-193 and CAST-32A. This report summarizes the activities we performed, focusing on how our verification activities addressed relevant AMC 20-193 and CAST-32A objectives. We also provide input to the DO-178C mandated deliverables, SAS and HAS.

3.2 Components of the solution

MACH178 includes the following components:

- **Platform and Software Analysis and Characterization Services**, which rely on the expertise of our specialist multicore engineers to understand and analyze multicore platforms and test multicore timing behavior against requirements.
- **Documents and tests**, template DO-178C planning documents for AC 20-193/ AMC 20-193 compliance, process documents and tests to support the solution.
- Rapita **Verification Suite** (R**VS**), a collection of embedded software verification tools that is widely used in the critical aerospace industry.
- Rapi**Daemons**, a collection of specialized programs to generate contention on shared hardware resources.
- **Integration** of R**VS** and Rapi**Daemons** with the multicore platform under analysis.
- **Certification support**, qualification kits and services to support DO-330 tool qualification of R**VS** and Rapi**Daemons**

3.2.1 Platform and Software Analysis and Characterization Services

To investigate the timing behavior of multicore software, our expert engineers study the multicore platform under analysis, elicit requirements, and use automated R**VS** tool support to run multicore timing tests to evaluate multicore hardware and produce timing evidence needed to meet DO-178C and AC 20-193/AMC 20-193 objectives.

3.2.2 Documents and tests

We provide template compliance documents for DO-178C certification in line with AC 20-193 and AMC 20-193 objectives, including a template Plan for Multicore Aspects of Certification which can be linked from the Plan for Software Aspects of Certification. We also provide process documents to support DO-178C compliance, and tests that support the analysis of multicore timing behavior.

3.2.3 Rapita Verification Suite

The following Rapita **Verification Suite** (R**VS**) plugins support **MACH**^{178}:

- Rapi**Test** automatically measures and reports execution time metrics from the task under analysis.
- Rapi**Time** lets engineers write multicore tests easily and automatically converts these into a test harness that checks software behavior.
- Rapi**Task** automatically measures and reports scheduling metrics for each task under analysis.

3.2.4 RapiDaemons

Rapi**Daemons** are specialized programs designed to generate contention on specific resources such as buses, caches and GPUs. Each Rapi**Daemon** is designed to either maximize or match a desired level of contention for a specific resource. As such, Rapi**Daemons** can be used to characterize hardware by investigating how different functions are affected by different types and levels of contention and quantify the impact of this contention (known as interference channels in AC 20-193/AMC 20-193).

After writing multicore timing tests with Rapi**Test**, Rapi**Daemons** are automatically applied to configure the desired level of resource contention in each test.

We verify the behavior of Rapi**Daemons** via requirements-based testing, ensuring that they execute as intended and stress the desired resources at the desired level. To verify them, we use independent metrics such as those provided by the Performance Monitoring Unit (PMU) on ARM-based chips.

Rapi**Daemons** are built on the Barcelona Supercomputing Center's microbenchmark technology (MuBT).

3.2.5 Integration

To support timing analysis on a multicore system, hardware and software components needed for the verification process must be integrated with the multicore platform under analysis.

This is achieved through an engineering service to integrate R**VS** plugins into the toolchain, put in place a suitable data collection mechanism, and port and configure any Rapi**Daemons** needed for the analysis.

3.2.6 Certification support

We provide qualification kits and services to support DO-330 tool qualification for R**VS** and Rapi**Daemons**, which are classed as Tool Qualification Level 5 tools.

4. Want to learn more?

If you want to learn more about multicore timing analysis, visit the Rapita Systems multicore timing discovery page at **rapitasystems.com/multicore-timing**. This includes videos, pre-recorded webinars and more informational resources on multicore timing analysis.

If you have a multicore system you need to verify timing behavior for, we'd love to hear about your project and see what we can do to help.

info@rapitasystems.com

We regularly release new material and run training courses on multicore timing analysis worldwide. To make sure you're notified, sign up to our mailing list.

rapitasystems.com/newsletter

About Rapita

Rapita Systems provides on-target software verification tools and services globally to the embedded aerospace and automotive electronics industries.

Our solutions help to increase software quality, deliver evidence to meet safety and certification objectives and reduce costs.

Find out more

A range of free high-quality materials are available at:
rapitasystems.com/downloads

SUPPORTING CUSTOMERS WITH:

Tools	Engineering Services	Multicore verification
Rapita **Verification Suite**:	V&V Services	**MACH**^{178}
Rapi**Test**	Integration Services	Multicore Timing Solution
Rapi**Cover**	Qualification	
Rapi**Time**	SW/HW Engineering	
Rapi**Task**	Compiler Verification	

Contact

Rapita Systems Ltd.
Atlas House
York, YO10 3JB
UK

+44 (0)1904 413945

Rapita Systems, Inc.
41131 Vincenti Ct.
Novi, Mi, 48375
USA

+1 248-957-9801

Rapita Systems S.L.
Parc UPC, Edificio K2M
c/ Jordi Girona, 1-3
Barcelona 08034
Spain

+34 93 351 02 05

rapitasystems.com

linkedin.com/company/rapita-systems

info@rapitasystems.com