MASTECS Multicore Timing Analysis on an Avionics Vehicle Management Computer

Raúl de la Cruz*, Philip Harris*, Samuel R. Thompson†, Christos Evripidou†,
Tim Loveless§, Juan M. Reina‡, Mikel Fernandez‡, Enrico Mezzetti‡, Francisco J. Cazorla‡
*Collins Aerospace Applied Research & Technology, Ireland
†Rapita Systems L.t.d, UK
§Lynx Software Technologies, UK
‡Barcelona Supercomputing Center, Spain

***Abstract*—Driven by the increasing compute performance required by modern autonomous systems, high-integrity applications are moving to multi-core processors as their main computing platform. Using multi-core processors in avionics is particularly challenging since the timing behavior of the software is not only affected by its inputs but also by software running simultaneously on other cores. To address this challenge the MASTECS project has developed a methodology for multicore timing analysis together with a supporting toolset. In this work we show the results of evaluating this methodology and tools on a representative avionics use case.**

***Index Terms*—Multicore Timing Analysis, Airborne Software, Robust Partitioning, CAST-32A**

I. Introduction

Autonomy features such as Advanced Air Mobility, Single Pilot Operation, and others, are driving commercial avionics systems towards multicore processors (MCPs) in pursuit of higher compute performance. MCPs are increasingly the central element of computing platforms for commercial avionics systems [15], [20]. As described in the CAST-32A position paper [6][1] and DOT/FAA/TC-16/51 report [11], significant MCP-related challenges such as software timing analysis have to be addressed before MCPs can be fully embraced. The key challenge with MCPs is the timing behavior of a piece of software is not only affected by its inputs but also by the software running simultaneously on other cores.

The development process of modern avionics systems is a very costly and time-consuming activity, taking as long as 5 years from requirements to the final certification [9]. To meet safety certification, the deterministic timing behavior of newly-developed avionics systems must be extensively proven to airworthiness authorities (e.g. FAA, EASA). Software certification is a time-consuming and largely manual process whose cost dwarfs the typical embedded development process. The long process to generate evidence exacerbates the burden of certifying MCP products for aerospace suppliers.

The MASTECS project [1] has developed a structured analytical approach (methodology and tools) to produce evidence about multicore timing behavior. The MASTECS test methodology is specifically designed to capture the impact of multicore contention on application behavior. It hinges on the use of micro-benchmarks (highly-targeted qualifiable interference generators) to simulate configurable resource contention, while using built-in performance monitoring functionality on the processor to capture application response. Tests are carried out on-target using the RapiTime timing analysis tool, with automation of test execution and analysis from the RapiTest tool. The test framework incorporates the capability to trace tests and results back up to specific verification goals.

In this work we present the application of the MASTECS Multicore Timing Analysis (MTA) methodology and some specific tools matured during the project to an avionics use case provided by MASTECS partner Collins Aerospace. The use case, which runs on an NXP T2080 platform with the LynxSecure Separation Kernel Hypervisor, is an adaptable-baseline DAL-A (flight-critical) Vehicle Management Computer (VMC) able to host third party and legacy applications. Specifically, we show the results of making an iteration of the MASTECS seven-step testing process based on a V-shaped verification model. This includes multicore Critical Configuration Settings (CCS), Interference Channels (ICH), and Hardware Event Monitor (HEM) analysis; identification of timing requirements; test case design; implementation of test procedures; evidence gathering (testing); results analysis; and results validation and generation of documentation.

The rest of this work is structured as follows: Section II introduces the commercial state of the art in MTA and introduces some relevant academic works. Section III develops the MASTECS approach to address MTA challenges. Section IV introduces the main elements of the MASTECS tool chain used in each step of the proposed methodology. Section V describes the roles of each partner during the project. Section VI introduces the use case and its timing requirements, shows how the toolchain has been applied to address those challenges, and show the results obtained. Section VII performs a retrospective TRL analysis on the MTA tooling and its potential application on industrial cases. Section VIII presents the main conclusions of this work.

II. Positioning

MCPs, along with other accelerators integrated on the same System on Chip, can in theory provide increased compute performance, power efficiency, and space efficiency, as required in future avionics systems. However, tasks executing on different cores can slow each other down due to competition

[1]In 2022, CAST-32A will be supplemented/superseded by AMC 20-193, a joint effort by EASA and the FAA. See Notice of Proposed Amendment [5].

for shared resources like caches, interconnection transactions and bandwidth, and hardware accelerator utilization. This situation is even worse when deriving worst-case execution time estimates as allowance must be made to account for worst case scenarios which are unlikely but feasible.

One of the challenges for MTA lies in quantification, demonstration, and documentation of the impact of multicore contention. This entails the definition of an analysis and testing approach that produces evidence of the timely execution of the software on the multicore platform. However, mitigation mechanisms can have a large overhead, so there is a delicate balance to be struck between demonstration of high determinism and detrimental loss of performance.

Recent works focus on providing a certification framework that helps applicants to prepare their CAST-32A certification activities [4]. The proposed framework uses graphical notation diagrams to organize the argumentation. It also proposes developing evidence via automated analysis.

Typical successful approaches to timing analysis of single-core systems are based on either static analysis or test-driven on-target measurement. However, scaling these to complex MCPs has proven challenging. The complexity of current and upcoming MCPs has been acknowledged to present a complexity wall for static timing analysis solutions [19]. On the software side, increasing complexity, for example to promote autonomous features [18], challenges structural and syntactical analyses. On the hardware side, hardware complexity and opaque IP conspire to render derivation of accurate timing models intractable. Measurement-based tools also present specific challenges for multi-core timing analysis [2]. Those are related to developing a methodology, producing the required evidence and traceability, and generation of stress scenarios to derive trustworthy timing bounds. In this line several works focus on improving platform observability. That is, they propose measurement environments with low-intrusiveness that allow collecting detailed information about event monitors with support for visualization [10], [12]. This requires tight interaction with the RTOS or the debug technologies available in the underlying hardware [12].

To the best of our knowledge, no available tool in the market provides these capabilities, namely, analyzing the timing behavior of applications running in a multicore and capturing the needs of safety standards and certification requirements.

III. MASTECS MTA Methodology

MASTECS' goal is to deliver an industrial MTA solution for safety-critical embedded systems. In this Section we introduce the MASTECS methodology for MTA and in Section IV how it is implemented via the MASTECS toolchain. Throughout this paper we use several terms that we introduce in Table I.

TABLE I: Terms, acronyms, and their definition.

Term	Definition
AMP	Asymmetric Multi-Processing
CCS	Crtical Configuration Setting
COTS	Common Off The Shelf
CSP	Certification Support Package
DMA	Direct Memory Access
GPU	Graphic Processing Unit
HEM	Hardware Event Monitor
ICH	Interference Channel
IFC	Intended Final Configuration
MCP	Multicore Processor
MPSoC	Multi-Processor System on Chip
MTA	Multicore Timing Analysis
SBC	Single Board Computer
SoC	System on Chip
QoS	Quality of Service
RVD	RVS Database
RVS	Rapita Verification Suite
TCM	Task Contention Model
TRM	Technical Reference Manual
VMC	Vehicle Management Computer
VM	Virtual Machine

A. Platform Analyses

Building a solid understanding of the underlying hardware platform is instrumental as the first step in any MTA project. It helps gain insight on the existing ICHs and potential mitigation actions that can be implemented to control the impact applications can suffer due to contention on those ICH.

Platform analysis mainly builds on the available Technical Reference Manuals (TRMs) for the board, the System on Chip (SoC), and any other I/O controller that can be used by the target application. Besides the TRMs, some hardware vendors provide Certification Support Packages (CSPs) that provide specific information useful from critical domains from more detailed descriptions of hardware blocks to hardware fault related information. It is noted that CSPs might not be free of charge, indeed they can be quite expensive, and require signing specific NDAs. Also some hardware vendors also provide consultancy support to solve specific questions that may arise during the platform analysis as long as it does not reveal any protected information on the hardware behavior.

There are three main elements to be identified in the analysis: CCS, ICH and HEMs.

CCS analysis. It aims to determine the set of platform control registers that hold configuration data such that an unintended modification can result in the hosted software not to comply with its functional, performance and timing requirements. The goal is to use mechanisms to protect them from unintended modifications or propose appropriate means of mitigation if CCS are inadvertently altered.

HEM analysis. It captures the observability of the platform's ICH. This step aims to determine the particular HEMs that help understand how applications exercise different ICH. These allow the measurement of the load an application or micro-benchmark puts on the ICH, to show whether an ICH is mitigated by means of specific measures that may be in place. It is worth mentioning that HEMs are used as a main building block to provide evidence that micro-benchmarks work as expected, i.e. to validate micro-benchmarks. However, HEMs should not be automatically trusted to work according to their described behavior in the corresponding TRM. Some work [3] already reports mismatches between the definition of some monitors in the corresponding TRMs and the values observed for specific experiments in the NVIDIA Jetson and Xavier MPSoCs, and the A53 in the Xilinx Zynq UltraScale+ MPSoC. Also errata documents [16] capture scenarios for the NXP iMX6 architecture in which certain performance monitors may not count events with precision. For the ARM A53 implementation in the Xilinx UltraScale+, some issues have also been reported with the instruction retired, store retired, and unaligned load/store retired event counters (among others) [21].

ICH analysis. MPSoC platforms in embedded critical domains already incorporate complex, high-performance, and Commercial Off-The-Shelf (COTS) hardware components including decentralized and distributed interconnects, deep shared cache hierarchies, DMAs, GPUs and other specialized, vendor-specific accelerators. Shared hardware resources are the root cause of multicore interference and so are the focus of timing analysis. ICH analysis builds on available technical information to first identify the main hardware shared resources in the platform (e.g. caches, interconnects, memories) and develop a description of the potential ICHs that exist in those shared resources.

The challenge lies in the fact that critical information for timing characterization is either not fully disclosed (to protect IP) or scattered across several documents. Furthermore, such data as is available can be relatively unclear and sometimes subject to errata. As a result, the list of potential ICHs identified have to be validated and characterized empirically.

Platform Analysis, i.e. CCS, HEM, and ICH analysis, drives the whole experimentation performed when following steps of the MTA process including the assessment of the impact CCS change on the application, the proposal of an ICH quantification plan, and the proposal of a HEM validation plan.

B. IFC Seclection

A smart selection of values for the identified CCS can both mitigate many ICHs while optimizing the performance of applications. As the hardware platforms become increasingly complex integrating more components, the number of CCS increases in every MPSoC generation. Also the variety of hardware features that can be controlled is wide. As representative examples, in the NXP T2080 [7], [8] the user can control the number of ways each core is allowed to use in the shared L2 cache, while the Xilinx Zynq UltraScale+ CCS allow control of quality of service (QoS) features that prioritize and route requests [17]. In the former case, deploying cache way partitioning prevents some of the ICHs in the L2 cache. In the latter request prioritization and routing prevents conflicts on the paths between different sources (e.g. computing elements application processing unit and the real-time processing unit) to a target (e.g. memory).

While intuitively cache partitioning, for instance, helps mitigate contention, it also can cause an application running in a core and confined to use a subset of the L2 cache to increase its number of L2 misses. As a result, the particular number of ways to assign to each core – which is configured via control registers (CCS) – depends on the particular application under consideration. Fixing CCS involves an iterative process in which several values are empirically evaluated until a good balance between isolation and performance is found. In MASTECS, we built on micro-benchmarks and the TCM to automate this process as covered in Section IV-C.

C. V Model

MASTECS methodology follows the standard V-model for software development life cycle with a well-defined set of analysis and test design activities, on the left side of the V, matched with corresponding verification and validation steps, on the other side. The fundamental steps in MASTECS MTA process model are summarized in Figure 1.

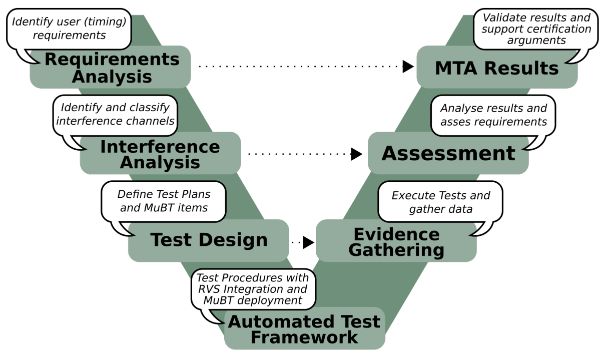
Identify user (timing) requirements
Requirements Analysis
Validate results and support certification arguments
MTA Results
Identify and classify interference channels
Interference Analysis
Analyse results and asses requirements
Assessment
Define Test Plans and MuBT items
Test Design
Execute Tests and gather data
Evidence Gathering
Test Procedures with RVS Integration and MuBT deployment
Automated Test Framework

Fig. 1: Steps proposed by MASTECS for MTA in the V-model software development process.

① Firstly, the user timing requirements are identified, which allows the remainder of the MTA process to be scoped appropriately. These can range from validating that a given cache partitioning mechanism prevent data evictions between applications to show that a micro-benchmark puts the desired level of load on a given ICH.

② The second step entails identifying ICHs through which interference between cores can take place. Also at this stage, the HEMs necessary for the analysis are identified, as are the CCS present in the platform. Note that Section III-A builds the knowledge on the potential ICHs in the platform, while this step specifically focuses on those ICH related to the timing requirement being addressed. It also leverages HEM analysis to restrict the analysis to the HEMs that capture the load on the ICHs identified as relevant.

③ The third step in the MASTECS methodology is to develop test cases to verify hypotheses supporting the user requirements, which includes defining the micro-benchmarks that will be used to exercise the ICH. Alongside ICHs, and HEMs, micro-benchmarks are the main elements in the test case argument.

The ④ fourth and ⑤ fifth steps focus on the implementation of the test procedures and their automated execute on the platform to gather test evidence. MASTECS exploits the Rapita Verification Suite (RVS) framework, from project partner Rapita Systems Ltd (RPT), to automate the execution of large batches of tests and the collection of raw information from the execution of the program under analysis on the real platform and configuration.

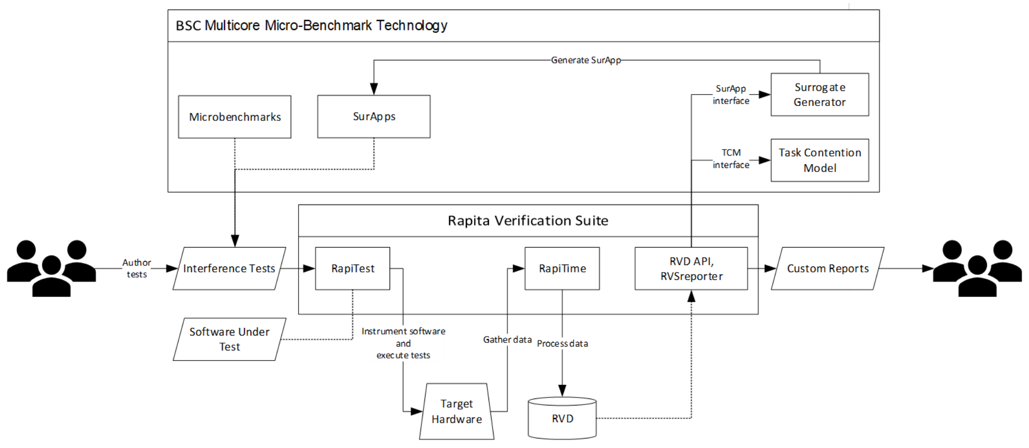

Fig. 2: MASTECS Toolchain.

In step ⑥, raw numbers are analyzed by technical expert to assess whether they prove (or otherwise disprove) the verification requirements. While the analysis step is only partially automated, it greatly benefits from RVS framework capability of providing different views and statistics of the gathered data.

The last step ⑦ involves a review of requirements, generation of certification artifacts to support the safety argument of the system. A characteristic feature of MASTECS MTA is that the whole analysis process is oriented towards fulfilling qualification and certification requirements as defined by domain-specific standards and regulations. Raw numbers and analysis results are presented as fundamental evidence to support a domain-specific certification arguments.

A key theme in the MASTECS MTA approach is the combination of the efforts of software timing analysis experts and hardware experts. This allows provision of the required insights into the behavior of modern complex MCPs running complex software. Hardware experts identify the ICH in the different hardware shared resources and any configuration options that may affect them – CCS according to CAST-32A. When it is determined than an ICH can be exercised by an application under test, hardware experts propose suitable HEMs to track contention in those ICH and a micro-benchmark design to put load on ICHs.

IV. MASTECS MTA Toolchain

The MTA toolchain supports the MTA methodology. It builds on the combination and integration of the RVS and Barcelona Supercomputing Center (BSC) Multicore Micro-Benchmark technology (MμBT) and multicore hardware knowledge. As shown in in Figure 2, Rapita's RapiTest lets the user to define the tests to carry out that usually involve the software under test (i.e. the application) and a micro-benchmark [2]. Rapita's RapiTime takes care of instrumenting the application according to user specification that captures the points of instrumentation and the specific HEMs to record at each point. The data collected from the execution is loaded in to an RVS Database (RVD). The RVS Exporter queries the RVD to provide information according to user's desired views. Information from the RVD is also provided as feedback on demand to BSC tools like the TCM and the Surrogate (Application) Generator that work iteratively.

[2]The Surrogate Applications (SurApps) are a type of micro-benchmarks that aim to copy the load that a reference application puts on the ICHs. In this work we do not assess SurApps.

A. Hardware Analysis

The hardware analysis process presented in Section III-A cannot be automated, that is, it is not possible to process TRMs to automatically extract CCS, HEM and ICH information. Hardware analysis is to be performed manually by hardware experts who should be providing insightful information about the hardware behavior.

Assessing the accuracy of the analysis is also difficult. Not only does it depend on the information made available to the hardware experts via the TRMs, CSPs, and consultancy support from the hardware provider, but also different hardware experts can produce slightly different conclusions in terms of ICHs. This risk can be mitigated by performing the analysis by different set of hardware experts which then combine their analyses into a single hardware analysis document encompassing CCS, ICH, and HEM analysis. It is also the case that robust guidelines on how to perform the analysis and reference analysis documents derived from previously analyzed processors can significantly help.

It is worth mentioning that, excepting the CSPs, none of the information used for ICH, HEM, and CCS analysis is intended for the purpose for which it is used in MTA. For instance, no section in the TRM captures ICH specifically. Instead, TRMs provide descriptions of how different hardware blocks interact. This description is provided to the level needed by software engineers to optimize the average performance of its applications or provide some QoS, as such, the TRM description is insufficient to provide all the details needed for ICH analysis.

This can be mitigated by performing a solid set of experiments to complement the analyses. HEMs are intended for general performance analysis and debugging purposes and usually lack descriptive information on exactly what events they track [3]. Previous experience and experimentation is needed to consolidate a set of trusted HEMs to use in the rest of the MTA process.

B. Multicore Micro-Benchmark Technology

MμBT is a suite of software tools that cover the low-level (hardware) aspects of MTA. In this section we cover micro-benchmarks, which are some of the main building blocks for MTA. Micro-benchmarks are single-behavior pieces of code that stress a specific ICH (shared resource), see Figure 3. By running the micro-benchmark against an application, one can assess the sensitivity and aggressiveness of the application to contention in a given ICH. Micro-benchmarks are specifically tailored to the hardware/software target configuration (IFC), and are a key tool to determine the bounds on the impact of ICHs and for assessing the effectiveness of interference mitigation techniques.

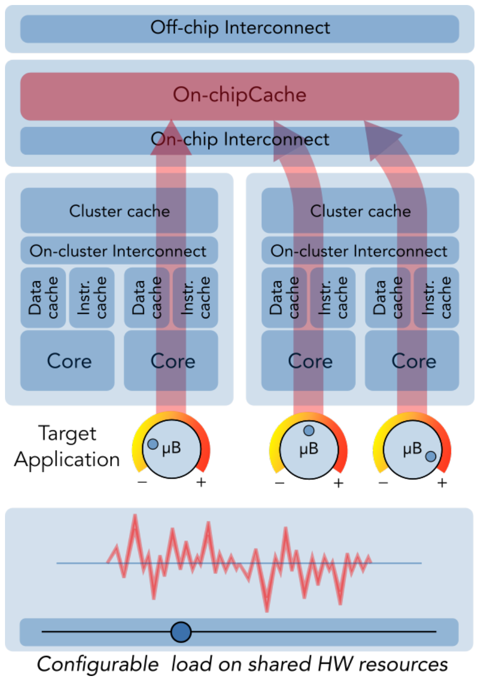

Fig. 3: Micro-benchmarks.

HEMs are used to assess the load micro-benchmarks put on ICHs and in general to validate the correct behavior of the micro-benchmark. Besides HEM-based validation a test harness is performed for the functional validation of each micro-benchmark under different scenarios.

PMUlib is a low-level library for configuring and reading of performance monitor counters serving as an access point to the available HEMs. It also supports an interface with tracing/debug units where present in processors (e.g. MultiCore Debug Solution in the Tricore AURIX TC39xx family) or external (Lauterbach). Reading HEMs can be performed in-band, i.e. from the software system under evaluation or via out-of-band debugger facilities [12] to prevent any probe effect.

C. Task Contention Model

The profiling information collected over the application in isolation can be conveniently exploited to provide early bounds on multicore timing interference incurred by the same application when deployed in a specific multicore scenario and under a given process schedule. The TCM exploits information on both the target HW and task model run-time to build a conservative analytical model for computing multicore contention. The model is not meant to provide exact estimates of multicore contention but early figures that can steer design and deployment decisions. The model, which is parametric on tasks' profile and schedule, allows fast exploration of any possible system configuration in view of reducing interference and optimizing the system makespan. The TCM aims at identifying a subset of candidate system configurations on which to focus the verification and validation efforts.

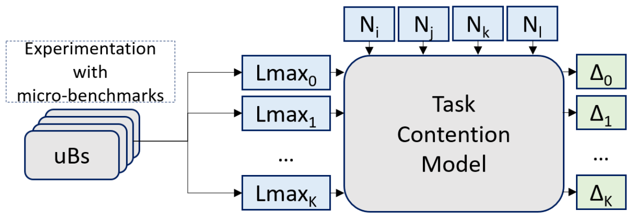

Fig. 4: Task Contention Model.

More in detail, the TCM builds on a timing estimate in isolation C_i^{isol} of a given task τ_i to derive a estimate of τ_i's execution time (C_i^{mcp}) when deployed in a specific multicore workload derived as $C_i^{mcp} = C_i^{isol} + \Delta_i$. The latter addend, Δ_i, is the composition of two elements, see Figure 4, (i) the longest contention different request types can suffer when accessing a shared resource r_i, $Lmax_i$,which is derived via execution of micro-benchmarks; (ii) the maximum number of requests, n_i, performed by τ_i and its contenders running in the other cores that are derived using PMULib (note that N_i is the maximum number of access of task τ_i to each resource, i.e. $N_i = \{n_{i,0}, n_{i,1}, ..., n_{i,k}\}$, where k is the number of shared resources. For instance, for a two task workload and one shared resource, the number of requests from τ_i that collide with co-runner τ_j in the access to the resource r_0 is defined as $min(n_{i,0}, n_{j,0})$. Hence, we can derive Δ_i as $min(n_{i,0}, n_{j,0}) \times Lmax_0$. The interested readers are referred to [13] for more information details on the fundamentals of the TCM.

D. RapiTest

RapiTest is a test-harness generator, capable of generating and driving both unit tests and system tests on-host and on-target.

For the MASTECS case studies, RapiTest was used to generate test harnesses and perform the necessary code injections to execute micro-benchmarks and collect timing data and HEMs as defined in the input interference tests.

RapiTest supports a range of native test input formats, in addition to automatic converters for a range of widely-used test formats. For the work documented here, RapiTest was configured to use two test formats designed explicitly for multicore testing. These formats allow simple description of locations at which micro-benchmarks should be injected to generate contention, the functions and call-trees that should be instrumented, and the data that should be collected at these locations.

E. RapiTime

RapiTime is an on-target timing measurement tool, which integrates static analysis of source code with on-target instrumentation for both timing and resource usage into a single hybrid timing analysis tool. Using RapiTime, it is possible to configure the automatic injection of instrumentation based on one or more pre-selected instrumentation profiles into certain functions, syntactic structures, or even whole call-trees. The instrumentation can be tightly controlled to minimise overhead, for example by only instrumenting certain locations, or by minimising the number of instrumentation points that make higher-overhead resource-usage measurements.

Many targets (including the T2080 used in this study) have a hardware limitation on the number of HEMs that can be collected at a time. To support this, RapiTime incorporates the concept of *metric groups*. A *metric group* is a selection of HEMs that can be collected simultaneously. If more than the maximum supported number of metrics is required, then RapiTime can split these resources into metric groups, repeat tests per metric group, and aggregate the data into a single report database.

RapiTime results are stored into an RVD database, which can be graphically interrogated using the RVS report viewer, or programmatically using the Python API. Custom text-based reports can be generated by the accompanying *rvsexporter* tool using a simple combination of Markdown and Python.

F. RVS Reporting Tools

All the RVS tools generate report files as RVD databases. To allow these databases to be interrogated, a few reporting tools are provided.

1) RVS Report Viewer: **RVS Report Viewer** features an interactive user interface that allows the user to explore the test results stored in an RVD database. For RapiTime reports, the report viewer gives access to (among other things):

- **Execution time:** Statistics for the maximum, average, minimum, and high watermark execution time for each instrumented function.
- **Contribution time:** The contribution to the overall execution time of the nominated roots made by each of the instrumented functions.
- **Invocation Timeline:** Per-invocation execution time data for each instrumented function.
- **Execution Time Profile:** Histograms showing the distribution of observed execution times for each instrumented function.
- **Metrics:** A range of visualizations for metrics collected from any available HEMs.
- **Coverage:** While RapiTime doesn't give the depth of information that RapiCover does, RapiTime instrumentation allows determination of which instrumented functions were executed and which were not.
- **Report Comparison:** Two reports containing measurements of the same thing can be compared. For example, a RapiTime report containing data from a test run when nothing was executing on the other cores of a system could be compared with another report when the same software was executed, but interference generators were running on the other cores.

From the RVS Report Viewer, it is possible to generate generic exports using the built-in default exporters. It's also possible to copy data tables directly into other software (e.g. a spreadsheet) for further analysis.

2) RVS Exporter: **RVS Exporter** is a means to generate custom report exports. RVS exporter takes an input template as a markdown file. This markdown report can contain embedded Python code that is evaluated when the template is processed by RVS Exporter. This embedded Python code has access to all the data in the results database, and the API also provides useful utility functionality to make accessing, processing, tabulating, visualising, and reporting on the data as simple as possible.

If some tests are repeated, it is a simple matter to run the RVS Exporter tool again to re-generate the report using the latest data. The modular structure of the reports also facilitates reuse of report fragments or processing algorithms between reports without unnecessary duplication.

V. MASTECS PARTNERS

The MASTECS consortium comprises two technology providers (BSC and RPT) and two end users (Collins Aerospace and Marelli Europe) in avionics and automotive who assessed the readiness the MASTECS MTA technology by evaluating it on their corresponding use cases. While it is not a partner of MASTECS, Lynx Software Technologies is also listed below as they contributed to the study presented in this work.

Barcelona Supercomputing Center (BSC) (Coordinator) is a leading research center in high-performance counting and embedded systems. BSC led the hardware analyses and matured the micro-benchmark technology, including the PMUlib and the TCM, to reach a high level of industrial readiness. In order to ensure a clear exploitation path of its technologies BSC created a spin-off company, Maspatechnologies S.L., during early stages of the project.

Rapita Systems Ltd. is a leading provider of software verification tools and services globally to the embedded aerospace and automotive electronics industries. Rapita has lead the productization of the technologies by developing tooling, processes, DO-178C documentation, tests and commercial infrastructure to bring a whole solution to an exploitable position within the market.

Collins Aerospace Applied Research & Technology (Collins-ART) is the innovation organization of Collins

Aerospace, a Raytheon Technologies company, leader in providing advanced solutions for the global aerospace and defense industry. Collins-ART has actively participated in MASTECS as end-user for the aerospace industry. Its main role was on setting avionics requirements; providing a representative aero case study; and demonstrating the effectiveness and soundness of the MTA toolchain during the evaluation.

Marelli Europe SpA. Marelli is one of the world's leading global independent suppliers to the automotive sector. With a strong and established track record in innovation and manufacturing excellence, Marelli's mission is to transform the future of mobility through working with customers and partners to create a safer, greener and better-connected world. Marelli has actively participated in MASTECS as end-user for the automotive industry. Its main role was on setting automotive requirements; providing a representative automotive case study; and demonstrating the effectiveness and soundness of the MTA toolchain during the evaluation.

Lynx Software Technologies specializes in real-time embedded safety-critical software. Lynx's contribution to the project was the LynxSecure product – a type 1 (bare-metal) hypervisor – as well as design, integration and support assistance. Such a hypervisor provides robust space partitioning without needing an RTOS, thus reducing HEM noise and allowing ICH (time partitioning) to be studied with higher fidelity. MASTECS used LynxSecure to partition the T2080 SoC's RAM, cores, peripherals and L2 cache hardware into bare-metal VMs.

VI. VMC Case Study

This section provides a summary of the case study evaluated including the platform where it runs, the particular instantiation of the MASTECS tool chain to cover the case study's requirements, and the results obtained.

A. *Introduction to the Case Study*

The case study builds on a redundant Flight Control System designed to replicate workload reduction applications at different levels of the system architecture: integration unit (VMC) and Single Board Computer (SBC). Each SBC board runs the system in an Asymmetric Multi-Processing fashion (AMP), and is able to deploy and run simultaneously mixed-criticality applications with different assurance levels (DAL-A/C).

Figure 5 shows the software architecture implemented for the VMC. Core 0 runs a process dedicated to I/O scheduling and data marshalling using FIFO queues and shared memory regions. The remaining cores of the SBC are dedicated to host applications accessing I/O through the queues provided by Core 0. Tasks are fully virtualized and executed on a Virtual Machine (VM) by the LynxSecure hypervisor, providing task domain isolation. The hypervisor allows the T2080's cores to be oversubscribed to host the 16 VMs of our case study. Each hosting core runs a VCPU manager (blue circle) that orchestrates and schedules computational tasks (green circles) in a pre-defined order to enforce data flow consistency. Lynx's Z-Scheduler is used on each VCPU manager to implement custom VM scheduling via time donation and as a convenient integration point for the HEMs and RapiTime tool.

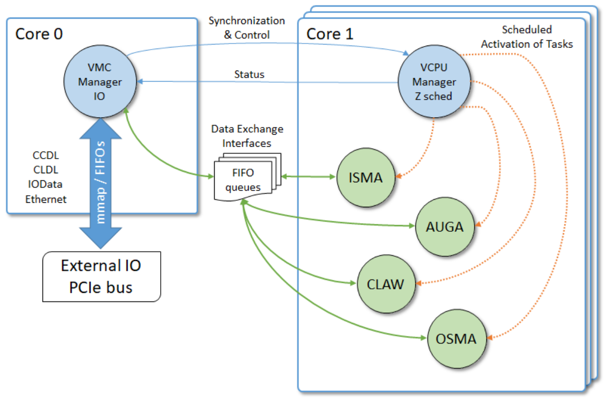

Fig. 5: Software architecture for each SBC of the VMC.

B. *The Target Platform*

The use case runs on an NXP T2080 processor [8], see Figure 6 which comprises 4 cores each its own private instruction and data cache. The L2 cache, CCF and DDR memory are shared among cores.

The LynxSecure hypervisor configures L2 cache using the T2080's hardware support for cache partitioning so that each core gets access to one fourth of the 16 cache ways (i.e. 4 ways). To that end the proper values are set to registers `L2PIRn`, `L2PARn`, and `L2PWRn`. In Figure 6 in the array in the L2 block, rows represent cache sets and columns represent cache ways.

Note that the CoreNet Coherence Fabric (CCF) is the main SoC interconnect and along with and the memory controller the focus for VMC case study. Interference caused by I/O activity is not included in this study.

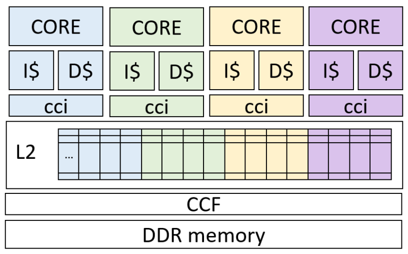

Fig. 6: Block Diagram of the main path from the cores to the DDR memory in the T2080. D$ and I$ stand for data and instruction caches, respectively; cci for core-cluster interface; and CCF for CoreNet Coherency Fabric.

C. *Tool Chain Instantiation*

Below we summarize how the MASTECS toolchain has been instantiated to address the VMC requirements.

① *Hardware analysis*: Hardware experts from BSC analyzed the T2080 processor [14]. Since the L2 cache is partitioned among cores it was concluded that it is not the main source of contention. The main sources are the CCF and main memory. Next, HEMs were identified to track activity on those resources and specific micro-benchmarks were designed to stress those resources. These include several counters in the L2 cache and in the Bus Interface Unit.

② *Verification requirements*: The particular requirements addressed in the scope of the evaluation include:

- REQ1. Determine whether idle cores may generate some noise: baseline time characterization experiments require a pristine configuration. Such default configuration (CCS) must ensure that no accesses are produced from unused devices to any shared resources.
- REQ2. Determine the overhead of HEMs reading: an accurate profiling of the tasks under analysis is fundamental to avoid incurring excessive probe effect and to discard any potential outliers in the computation of WCETs. The LynxSecure hypervisor, the PMUlib, and RVS components are assessed and configured to provide accurate instrumentation.
- REQ3. Assess the increase in execution time and HEM values due to contention triggered by well-designed micro-benchmarks: for this purpose, the taskset must be instrumented and monitored both in isolation, to capture application timing baseline, and under stress scenarios where multicore timing interference arises.
- REQ4. Obtain early estimates of the impact of multicore contention on the application timing behaviour: the TCM shall allow the generation of task scheduling schemes where interference and makespan are reduced.

③ *Test Cases*: In order to capture REQ1 and REQ2 we designed test cases in which the task under analysis is run in isolation. REQ3 and REQ4 also require experiments in multicore scenarios, which specific micro-benchmarks running in different cores with or without the application under analysis.

④ Test procedures and their execution: an executable test procedure is generated for each test case allowing automated execution of the test cases and collection of the results.

⑤-⑥ Test Results: The raw results are analyzed to assess verification requirements incrementally. Results for REQ1 let us assess whether idle cores generate noise due to any background activity. REQ2 results enable calibration that the HEMs readings are accurate and a trustworthy building block for MASTECS analyses. Finally evidence for REQ3 and REQ4 provides insight on the impact of contention.

D. Results

In the following we report the results from applying MASTECS technology to fulfill verification requirements REQ1-4. All experiments build on the use of RVS tool to collect timing and relevant hardware events on the final VMC hardware and software configuration. The RVS tool gathers information at the desired granularity whilst the program under analysis executes. In the scope of this case study, we instructed the tool to automatically insert software instrumentation points for all 16 processes. Since each process consists of distinct Read, Computation and Write steps, information is collected at the granularity of each step.

1) REQ1: In order to fulfill REQ1 we prepared an experiment using the same same VM workload configuration used in the final setup and a much simpler experimental setup in which we execute a single micro-benchmark in one of the cpu cores while the remaining ones are left idle. We run several micro-benchmarks:

- RO1B. Micro-benchmark accessing and hitting one bank of the L2 with read operations.
- WO1B. Micro-benchmark accessing and hitting one bank of the L2 with write operations.
- RO4B. Micro-benchmark accessing and hitting in all banks of the L2 with read operations.
- WO4B. Micro-benchmark accessing and hitting in all banks of the L2 with write operations.
- RO. Micro-benchmark accessing and missing in the L2 and going to memory where it generates read operations.
- RW. Micro-benchmark accessing and missing in the L2 and going to memory where it generates read and write operations.

We leverage the per-core (per-thread) counters in the L2 cache. By comparing the per-core, also known as local, HEMs in the L2 with global counters we can assess whether the other cores are generating additional activity. In particular, we read the following global/per-thread hardware event pairs: L2 hits (456-global and 465-local), L2 misses (457-global and 466-local), L2 store allocates (460-global and 468-local), and L2 data accesses (462-global and 470-local). In Table II we present the results of the differences between each pair of global and per-thread HEMs. As it can be seen the local and global activity matches quite well which shows that the only activity generated in the cache is that coming where the micro-benchmark runs. The differences between global and per-thread HEMs are lower than 0.69%, which shows that noise from idle cores is negligible in the tested configuration.

TABLE II: Ratio between global and per-thread HEM pairs.

HEM	L2Hit	L2Miss	L2StAlloc	L2DataAcc
RO1B	0.20%	0.01%	0.00%	0.20%
WO1B	0.45%	0.13%	0.05%	0.56%
RO4B	0.03%	0.00%	0.00%	0.03%
WO4B	0.42%	0.29%	0.15%	0.68%
RO	0.21%	0.20%	0.06%	0.40%
RW	0.42%	0.30%	0.16%	0.69%

2) REQ2: To fulfill REQ2, we instructed RVS to collect execution information on timing, instructions, and memory accesses through local (per thread) and global (per platform) hardware counters. In particular we instrumented a dummy function on which we expect no activity and assessed RVS+PMULib instrumentation overhead against a reference scenario with minimal HEM manipulation (PMU only). At the beginning and end of the function we read 6 on-core HEMs gathered on the T2080 platform during the profiling experiments. These are per-core HEMs Processor cycles (CYC, 001), Instructions completed (INS, 002), SGB promotions (SGBP, 230), DLINK requests (DLINKR, 443) that are per core HEMs; and the L2-related HEMs L2 misses per thread

(L2Mt, 466), L2 store allocates per thread (L2STAt, 468), L2 Data accesses per thread (DL2At), and L2 Data misses per thread (DL2Mt). Those were specifically selected as they provide information on the instruction mix with emphasis on the memory operations, requests to the DL1, L2, and memory.

Table III shows the values observed. With manual instrumentation using PMULib on top of LynxSecure we observe minimal instruction overhead and no memory request. With the automation provided by the RVS infrastructure we observe very low absolute values that become negligible in relative terms as soon as the instrumented function is in the order of hundred of thousands of cycles, translating into micro-seconds.

TABLE III: Probe effect analysis. Instrumentation noise using PMULib on top of different setup layers.

HEM	CYC	INST	SGBP	DLINKR	L2Ht	L2Mt	L2STAt	DL2At	DL2Mt
ID	1	2	230	443	465	466	468	470	472
PMULib	21	5	0	0	0	0	0	0	0
RVS	1129	1562	85	85	71	2	3	57	4

3) REQ3: In order to capture REQ3 we use RapiTime to generate WCET estimates for each processes under both isolation (cycles solo) and contention scenarios with the RO and RW micro-benchmarks (RO sld and RW sld, respectively). This involves executing tasks against tailored, memory-aggressive micro-benchmarks, which are automatically stubbed by RVS. We also report the slowdown captured by RVS when the process is executed in the IFC, that is, without micro-benchmarks and with the other processes running in parallel (Parallel sld). Results are reported in Table IV.

We observe that the slowdown generated by the RO micro-benchmark is generally low. In fact, most of the times, the suffered interference is smaller than that observed in the IFC (see Observed slowdown in Table V). The contention impact of the RW micro-benchmark, instead, is always higher than taht in the IFC, effectively upper bounding it.

In general, the slowdown generated by the micro-benchmark is limited for the processes lasting longer and vice-versa. At the extremes of the spectrum we find PROC6 that lasts millions of cycles (10e6) and suffers a slowdown of only ∼1.10 for both core1 an core2; and PROC8 that lasts dozens of thousands of cycles (10e4) and suffers slowdowns around 5.5x when is contended against RW. As PROC8 has high density of access to memory, it suffers high slowdown against the RW micro-benchmark. However, PROC8 seems not to suffer contention from the other running processes in the IFC. Finally, all processes with a duration in the order of hundreds of thousands of cycles (10e5) display slowdowns that range from 1.40 to 3.00x against the RW.

4) REQ4: A TCM tailored to VMC hardware and software configuration has been developed and assessed in MASTECS. In particular, the tool has been integrated together with a scheduling and mapping optimization framework to provide an early assessment of different deployment configurations and identify those schedule scenarios that are not jeopardized by multicore timing interference.

TABLE IV: Core 1 and 2 results under contention scenarios. A single-core is activated with the process under analysis along with RO and RW micro-benchmarks on Core 3.

	Core 1			Core 2		
Process ID	Cycles in isolation	RO slowdown	RW slowdown	Cycles in isolation	RO slowdown	RW slowdown
PROC1(10e5)	733925	1.01	2.63	737283	1.01	2.57
PROC2(10e5)	370678	1.00	1.40	371201	1.00	1.40
PROC3(10e5)	746765	1.00	2.45	796911	1.00	2.41
PROC4(10e5)	160812	1.05	3.00	159425	1.05	2.98
PROC5(10e5)	736484	1.01	2.12	748450	1.01	2.06
PROC6(10e6)	3785988	1.00	1.10	3786251	1.00	1.09
PROC7(10e5)	740612	1.01	2.47	749216	1.00	2.44
PROC8(10e4)	57404	1.16	5.52	56599	1.16	5.51

In the following we evaluate the TCM bounds on an example deployment scenario and schedule. The profiling information on the VMC processes in isolation has been fed to the TCM and the obtained analytical bounds on multicore timing interference are assessed against maximum observed slowdown in real experiments.

TABLE V: TCM bounds against observed values.

	Core 1			Core 2		
Process ID	Time in Isolation	Observed Slowdown	TCM Bound	Time in Isolation	Observed Slowdown	TCM Bound
PROC1	733925	1.06	1.06	737283	1.07	1.06
PROC2	370678	1.01	1.17	371201	1.00	1.17
PROC3	746765	1.00	1.06	796911	1.00	1.06
PROC4	160812	1.06	1.67	159425	1.06	1.66
PROC5	736484	1.06	1.06	748450	1.07	1.06
PROC6	3785988	1.02	1.06	3786251	1.03	1.06
PROC7	740612	1.05	1.06	749216	1.06	1.06
PROC8	57404	1.03	2.62	56599	1.02	2.65

Table V reports the (maximum) observed and computed relative slowdown suffered by each VMC process because of contention. Results show the TCM results are generally upper-bounding the impact of contention, modulo a ∼1% tolerance threshold due to unaccounted negligible activity on the VMC Manager. While bounds are generally tight, in few cases the TCM bound seem to be overly pessimistic (see PROC4 and PROC8 in both cores). It should be noted, however, that pessimism is only apparent as it is generally difficult to hit the worst-case contention scenthose cases, namely PROC8 in both cores, correspond to short, memory intensive tasks (with ∼10% of instructions being memory accesses) where relative impact of memory accesses is extremely large and so is the maximum impact of contention, which is not easy to trigger with simple observations.

VII. Perspective

The MASTECS project partners have achieved success in bringing the technologies to a good commercial position and technical maturity, providing a foundation for deployment of the MASTECS methodology in support of certification of emerging aircraft systems.

The key to achieving industry-wide benefit from these tools and techniques is to ensure that the technology and commercial models enable the manufacturers and users to build on and share best practice. For example, using a standardized process that is familiar to certification authorities reduces risk of failing to achieve certification, repeatable automation abstracts from the challenges of low-level testing making it economic, and reusable IP designed and tested/qualified for use in high-integrity systems means a faster time to market.

The automotive and aerospace case studies were highly valuable in providing feedback to the technologies and guiding the process of "productization", helping to steer the technology partners in meeting the needs of real aerospace and automotive projects - this is an example of sharing best practice to benefit the whole industry.

The significant challenges of building safety-critical systems on multicore technology will continue. As new platforms appear with new performance enhancing features (such as multi-level caches, DMA, decentralized interconnects, GPU and other accelerators) the technology required to support them will continue to develop too, building on the baselines in this paper. Expect many more developments in this area.

VIII. Conclusions

The pursuit of increased performance in critical domains is relentless, and the avionics domain is not an exception. Advanced increased-autonomy related features like Advanced Air Mobility and Single Pilot Operation, require unprecedented levels of computing performance. The use of multicore processors is the main path followed to provide the required performance. The other side of the coin is that multicores bring their own challenges including software timing analysis. In this work we have presented the MASTECS Multicore Timing Analysis methodology and tools. We also showed its application to an avionics case study. Both help assessing how MASTECS technology helps achieving CAST32-A/A(M)C20-193 requirements.

Acknowledgments

This work has been supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GBC21/AEI/10.13039/501100011033 and the European Unions Horizon 2020 Framework Programme under grant agreement No. 878752 (MASTECS).

References

[1] MASTECS: Multicore analysis service and tools for embedded critical systems. https://mastecs-project.eu/.

[2] Jaume Abella, Carles Hernández, Eduardo Quiñones, Francisco J. Cazorla, Philippa Ryan Conmy, Mikel Azkarate-askasua, Jon Pérez, Enrico Mezzetti, and Tullio Vardanega. WCET analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015, Siegen, Germany, June 8-10, 2015*, pages 39–48. IEEE, 2015. doi:10.1109/SIES.2015.7185039.

[3] Javier Barrera, Leonidas Kosmidis, Hamid Tabani, Enrico Mezzetti, Jaume Abella, Mikel Fernández, Guillem Bernat, and Francisco J. Cazorla. On the reliability of hardware event monitors in mpsocs for critical domains. In Chih-Cheng Hung, Tomás Cerný, Dongwan Shin, and Alessio Bechini, editors, *SAC '20: The 35th ACM/SIGAPP Symposium on Applied Computing, online event, [Brno, Czech Republic], March 30 - April 3, 2020*, pages 580–589. ACM, 2020. doi:10.1145/3341105.3373955.

[4] Frederic Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, and Alfonso Mascarenas Gonzalez. PHYLOG certification methodology: a sane way to embed multi-core processors. In *10th European Congress Embedded Real Time Systems, ERTS 2020, Jan-Feb 2020*, 2020.

[5] European Union Aviation Safety Agency. Notice of Proposed Amendment 2020-09. https://www.easa.europa.eu/sites/default/files/dfu/npa_2020-09_0.pdf, 2020.

[6] Federal Aviation Administration, Certification Authorities Software Team (CAST). *CAST-32A Multi-core Processors*, 2016.

[7] Freescale semicondutor. e6500 Core Reference Manual. https://www.nxp.com/docs/en/reference-manual/E6500RM.pdf, 2014. E6500RM. Rev 0. 06/2014.

[8] Freescale semicondutor. QorIQ T2080 Reference Manual, 2016. Also supports T2081. Document Number: T2080RM. Rev. 3, 11/2016.

[9] Scott Gerhold, Mike Dunham, and Branden Sletteland. Alternative multi-core processor considerations for aviation. In *AHS International 74th Annual Forum & Technology Display, Phoenix, Arizona, USA, May 14-17, 2018*, 2018.

[10] Sylvain Girbal, Jimmy Le Rhun, and Hadi Saoud. METrICS: a Measurement Environment For Multi-Core Time Critical Systems. In *9th European Congress Embedded Real Time Systems, ERTS. Jan-Feb 2018*, 2018.

[11] Laurence H. Mutuel, Xavier Jean, Vincent Brindejonc, Anthony Roger, Thomas Megel, and E. Alepins. Assurance of multicore processors in airborne systems. DOT/FAA/TC-16/51, Federal Aviation Administration, 2017.

[12] Xavier Palomo, Mikel Fernández, Sylvain Girbal, Enrico Mezzetti, Jaume Abella, Francisco J. Cazorla, and Laurent Rioux. Tracing hardware monitors in the GR712RC multicore platform: Challenges and lessons learnt from a space case study. In *32nd Euromicro Conference on Real-Time Systems, ECRTS 2020, July 7-10, 2020, Virtual Conference*, volume 165 of *LIPIcs*, pages 15:1–15:25, 2020.

[13] Xavier Palomo, Enrico Mezzetti, Jaume Abella, Reinder J. Bril, and Francisco J. Cazorla. Accurate ilp-based contention modeling on statically scheduled multicore systems. In *25th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2019, Montreal, QC, Canada, April 16-18, 2019*, 2019.

[14] Roger Pujol, Hamid Tabani, Jaume Abella, Mohamed Hassan, and Francisco J. Cazorla. Empirical evidence for mpsocs in critical systems: The case of NXP's T2080 cache coherence. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1162–1165, 2021. doi:10.23919/DATE51398.2021.9474078.

[15] David Radack, Harold G. Tiedeman, and Paul Parkinson. Civil certification of multi-core processing systems in commercial avionics. Technical report, Rockwell Collins, 2018.

[16] NXP Semiconductors. Chip Errata for the i.MX 6SLL. Document Number: IMX6SLLCE, 2019.

[17] Alejandro Serrano-Cases, Juan M. Reina, Jaume Abella, Enrico Mezzetti, and Francisco J. Cazorla. Leveraging hardware qos to control contention in the xilinx zynq ultrascale+ mpsoc. In Björn B. Brandenburg, editor, *33rd Euromicro Conference on Real-Time Systems, ECRTS 2021, July 5-9, 2021, Virtual Conference*, 2021.

[18] Hamid Tabani, Leonidas Kosmidis, Jaume Abella, Francisco J. Cazorla, and Guillem Bernat. Assessing the adherence of an industrial autonomous driving framework to ISO 26262 software guidelines. In *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*, page 9. ACM, 2019. doi:10.1145/3316781.3317779.

[19] Reinhard Wilhelm. Mixed feelings about mixed criticality (invited paper). In Florian Brandner, editor, *18th International Workshop on Worst-Case Execution Time Analysis, WCET 2018, July 3, 2018, Barcelona, Spain*, volume 63 of *OASICS*, pages 1:1–1:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/OASIcs.WCET.2018.1.

[20] Frank Wolfe. EASA and FAA to issue further guidance on multicore certification this year. *Aviation Today*, February 2020. URL: https://www.aviationtoday.com/2020/02/28/easa-and-faa-to-issue-further-guidance-on-multicore-certification-this-year/.

[21] Xilinx. Zynq UltraScale+ MPSoC, APU - PMU Counter Values Might Be Inaccurate When Monitoring Certain Events. Document Number: AR# 68878, 2017.