Industrial Challenge 2022: A High-Performance Real-Time Case Study on Arm

Matteo Andreozzi
Arm, Cambridge, United Kingdom

Giacomo Gabrielli
Arm, Cambridge, United Kingdom

Balaji Venu
Arm, Cambridge, United Kingdom

Giacomo Travaglini
Arm, Cambridge, United Kingdom

Abstract

High-performance real-time systems are becoming increasingly common in several application domains, including automotive, robotics, and embedded. To meet the growing performance requirements of the emerging applications, these systems often adopt a heterogeneous System-on-Chip hardware architecture comprising multiple high-performance CPUs and one or more domain-specific accelerators. At the same time, the applications running on these systems are subject to stringent real-time and safety requirements. Due to the non-deterministic execution model of the compute elements involved and the co-location of the workloads, which leads to contention of the shared hardware resources, designing and orchestrating such applications is particularly challenging. In fact, the demand for novel methodologies, tools, and best practices to assist application designers working on high-performance real-time systems has never been stronger.

To stimulate innovation in this area, this document outlines an industrial case study from the automotive domain targeting an Arm-based hardware platform. The selected application is an augmented reality head-up display, which can be considered a representative example of a high-performance real-time use case. This case study will serve as the basis for a (multi-year) challenge involving real-time and embedded systems researchers across academia and industry that will be kicked off at the 34th Euromicro Conference on Real-Time Systems (ECRTS) 2022.

2012 ACM Subject Classification Computer systems organization → Real-time systems; Computer systems organization → Embedded and cyber-physical systems; Computing methodologies → Modeling and simulation

Keywords and phrases real-time, worst-case execution time

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2022.1

Acknowledgements The authors would like to thank Jay Cha (Arm) for his contribution to the definition of the case study, and Sophie Quinton (Inria), Martina Maggio (Saarland University) and Marko Bertogna (University of Modena) for their insightful comments and suggestions.

1 Introduction

As computing becomes ubiquitous, we observe increased interactions between devices and the physical world, which implies dealing more often with *real-time* and *safety* requirements. At the same time, the growing complexity of the end applications and the amount of data to be processed contribute to raising the *performance* requirements of the compute devices. These two trends lead to the proliferation of *high-performance real-time systems*.

Such systems are usually characterized by the co-location of multiple workloads on a single compute substrate, typically a heterogeneous System-on-Chip (SoC). This is done to improve the overall utilization of system resources by enabling their reuse (e.g., IO devices, hardware accelerators, etc.), and to improve the efficiency of data sharing across workloads. Workloads executing on such systems are defined as *mixed-criticality* [14]: each of them requires a potentially different *Level of Service* from the system, e.g., requiring a deadline to be met, a certain memory latency not to be exceeded, or bandwidth of requests being satisfied by a compute unit such as a GPU, with varying consequences, from soft recoverable errors to catastrophic failures, if those requirements are not met by the system at any point in time.

Co-locating multiple workloads with varying levels of criticality and priority comes at the cost of potential performance degradation due to the risk of interference between these workloads executing on shared resources, and thus increases the complexity of real-time analysis. The challenge in designing mixed-criticality systems is primarily to guarantee sufficient partitioning/isolation while still achieving high performance. Hence, it becomes necessary to be able to provision resources in a quantifiable and predictable way, regardless of whether the execution time of a workload may have non-deterministic external dependencies, such as non-deterministic data values or the arrival of non-deterministic events (e.g., interrupts). This is crucial for computing the *Worst-Case Execution Time* (WCET) for the real-time workloads being executed on the platform, and to ensure smooth and responsive operation of the general-purpose operating system (GPOS) workloads.

Addressing the above issues in a comprehensive way is one of the biggest challenges faced by system and application designers across various domains, in particular in the automotive, robotics, and Internet of Things (IoT) sectors. For this reason, we believe that it is very important to stimulate research across industry and academia around high-performance real-time themes. In this context, we would like to introduce the real-time research community to the Industrial Challenge associated with the Euromicro Conference on Real-Time Systems (ECRTS). Based on the success of the past editions of the challenge, which were part of the Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), a satellite workshop of ECRTS, we plan to follow a similar format: the challenge participants will be asked to address a specific set of initial questions, targeting approximately a 1-year timeline, and, based on the interest of the community and the reception of the initial set of activities, we will propose additional, more advanced, activities to be addressed in the following years. This document provides an outline of the scope and activities that we envision for the challenge. We propose an augmented reality head-up display application as a motivating case study, which we hope will provide an interesting testbed for innovative approaches in the areas of tools, methodologies and best practices to analyze high-performance real-time systems.

While the description of the case study provided in this document is a good starting point for the groups willing to start working on the challenge, we anticipate that further refinements to the definition of the case study and/or to the challenge activities will be released in the next months, together with deliverables including simulation tools, profiling data and reference input sets for the various software tasks. The web page `https://www.ecrts.org/arm-industrial-challenge/` will be used to share such updates and deliverables. If participants will have additional questions, we encourage them to submit them to the `#industrial-challenge` Discord channel – we will monitor the channel and address those questions in a timely fashion.

In the remainder of this document, Section 2 will describe the case study considered in detail. Section 3 will cover the key activities of the challenge, including a description of the related work; while some of the related work is mainly pertinent to the initial set of questions addressed to the challenge participants, we cover other related work that could be relevant to follow-up questions, in order to expand the scope of the challenge. Section 4 will present the recommended platforms for the evaluation of the case study. The resources that will be provided to the challenge participants, which include analysis tools, pointers to the recommended software implementations for the main application tasks, and profiling data, will be covered in Section 5. We will conclude with final remarks in Section 6.

2 Case Study

The case study selected for the Industrial Challenge 2022 is an augmented reality head-up display application (AR HUD) for the automotive market, appropriately simplified to allow the study of its real-time aspects within the anticipated timeline and scope of the challenge.

AR HUDs extend the exterior view of the traffic conditions in front of the vehicle with virtual information (augmentations) for the driver. They are used to improve the situational awareness of drivers by displaying graphics that interact with the driver's field-of-view (FOV). The information provided is generated from real-time sensor data and typically includes advanced driver assistance system (ADAS) alerts and navigational cues overlaid on real-world objects. AR HUDs have started to appear in high-end cars and are expected to become a relatively common feature in the future due to the safety and comfort enhancements that they bring to the driving experience. At the same time, given their demanding compute and real-time requirements, AR HUDs are good examples of high-performance real-time applications where the interactions between the software tasks and the utilization of the shared hardware resources need to be carefully orchestrated to meet the desired functionality and performance goals.

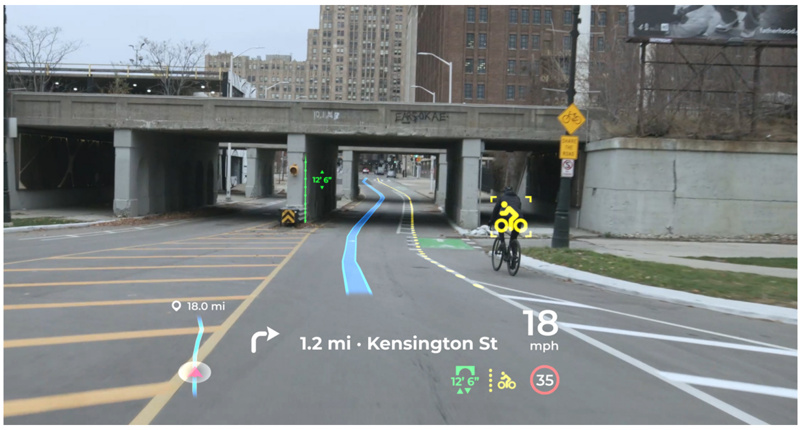


Figure 1 An example of AR HUD (from [13]).

A key requirement for AR HUDs is the ability to project the images with enough positional accuracy to create the illusion that they appear as "fused" with the real world. This can be particularly challenging in driving scenarios due to the rapidly changing environment and the amount of sensor data to process and to present to the driver. In addition, to adjust the image to the driver's viewpoint, an eye tracking or head pose estimation function is normally used, which determines the appropriate amount of rotation/distortion to apply to the image frame. Another important requirement for AR HUDs is the ability to project images at a sufficient distance in front of the driver (around 10m): projecting images at longer distances reduces the accommodation time for the eyes between the real world and the HUD images; in addition, the ability of the human vision system to distinguish depth from other real-world objects diminishes greatly beyond 7m – this factor can have implications on the complexity of the required eye tracking/head pose estimation function. Finally, high frame rates are necessary to avoid negatively impacting the user experience.

The implementation of an AR HUD system requires several functions, which can be decomposed into different software tasks. Figure 2 shows those functions, their decomposition into software tasks and a hypothetical mapping of such tasks onto the SoC compute resources (please refer to Section 2.5 for more details on the target hardware platform). The decomposition shows that we have a mix of real-time critical tasks and high-priority tasks, which can be mapped onto different concurrent CPU threads, thus making this case study non-trivial for real-time analysis. The *head pose estimation* task, implemented through a neural network, benefits from being mapped onto a high-throughput accelerator, either a GPU or a dedicated Neural Processing Unit (NPU). The following subsections will provide a detailed description of the tasks.

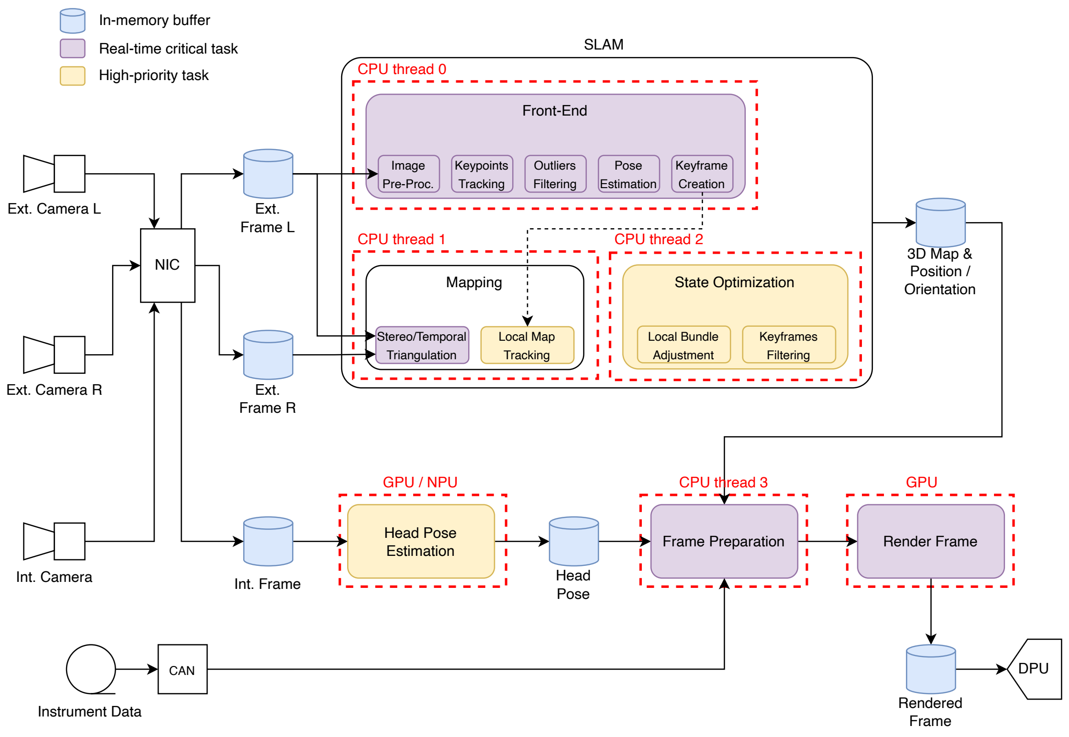


Figure 2 Software tasks comprising the AR HUD case study and their hypothetical mapping onto the SoC compute resources. Purple blocks represent real-time critical tasks, while yellow blocks represent non-critical, high-priority, tasks. Red contours highlight the mapping to the compute blocks, i.e., CPU threads and GPU/NPU tasks. The arrows indicate the high-level data flow.

2.1 SLAM

A Visual Simultaneous Localization and Mapping (SLAM) function is required to determine the orientation and trajectory of the vehicle and to generate a map of its surroundings. The output of this function is then used to generate the HUD graphic images and to position them within the driver's FOV.

The SLAM implementation selected for this case study is based on OV^{2}SLAM [5], a high-performance feature-based SLAM supporting both monocular or stereo camera setups. While other SLAM techniques can offer improved accuracy, OV^{2}SLAM is one of the techniques with a publicly available implementation that aims at supporting real-time performance for a variety of realistic scenarios, (e.g., including autonomous driving), and can trade-off accuracy to maintain the required real-time performance.

For this case study, we assume a stereo camera setup: the SoC is connected to a pair of external cameras through Automotive Ethernet. The cameras have High-Definition (HD) resolution (1920x1080 pixels) and frame acquisition is expected to be performed at a relatively high target rate, in the range 30-60 frames per second (FPS). A Network Interface Card (NIC) receives new frames from the cameras and deposits them into a memory buffer, so that they can be read by the consumer tasks.

The *front-end* task includes the following sub-tasks:

- *image pre-processing*: a contrast enhancement technique is applied to all new frames to increase the dynamic range. Dynamic range in photography describes the ratio between the maximum and minimum measurable light intensities. Bright parts of the image can get much brighter, so the image seems to have more "depth" aiding the following stages of processing. The algorithm also limits the intensity changes due to exposure adaptation as the car drives through bright and shaded regions;
- *keypoint tracking*: an optical flow algorithm is applied to determine the *keypoints* and their motion, based on a pyramidal implementation of the inverse compositional Lucas-Kanade (LK) algorithm; keypoints are interesting portions of the images (eg: corners of objects in the image) that are tracked in consecutive frames. The number of keypoints are generally configurable in the algorithm.
- *outlier filtering*: outlier keypoints are identified by applying RANSAC filtering based on the epipolar constraint and removed in order to improve the accuracy of the camera pose estimation;
- *pose estimation*: this is performed by minimization of the 3D keypoints reprojection errors using a robust Huber cost function;
- *keyframe creation*: if the number of tracked 3D keypoints (i.e. the keypoints with prior information on their real 3D position) w.r.t. the last keyframe gets under a threshold or if a significant parallax is detected, a new keyframe is created. The scenario of number of tracked 3D keypoints falling below a threshold when compared to 3D keypoints in the last keyframe occurs when there is drastic changes in the scene while driving. This will be detected and a new keyframe needs be created for the new scene.

More details for these sub-tasks are available in [5]. The front-end pipeline is fully monocular, limiting all its operations to frames provided by the left camera, even if a stereo setup is available.

The *mapping* thread is responsible for processing every new *keyframe* to create new 3D map points by triangulation (both *stereo & temporal triangulation* with a stereo camera setup) and to perform *local map tracking* in order to minimize drift. These two sub-tasks have different real-time requirements: triangulation needs to operate at the full frame rate as it is critical for keeping accurate pose estimation in the front-end; the local map tracking operation, on the other hand, does not need to run at the full frame rate and it is executed and aborted if a new keyframe is available. However, it is beneficial to keep the local map tracking task as a higher priority task than, for instance, general-purpose or background tasks, as its frequency of execution has an impact on the overall SLAM accuracy.

The *state optimization* thread is responsible for running a *local bundle adjustment* (BA) pass, that is applied to refine the poses of the most recent keyframes and 3D map points' positions, and a *keyframes filtering* pass, that is applied to filter redundant keyframes in order to reduce the runtime of future instances of BA.

As with most high-performance real-time SLAM implementations, OV^{2}SLAM leverages multi-threading to achieve real-time performance. As highlighted in Figure 2, the considered implementation of OV^{2}SLAM relies on three CPU threads.

While OV^{2}SLAM supports loop closure, this feature is disabled for this case study: due to the nature of the AR HUD application, the construction of a global map is, in fact, largely unnecessary. Loop closure and global map construction are more useful in scenarios where the user will revisit the same region of the map, e.g., for an AR wearable use case. However, while driving, this is more of a rare scenario and hence global map construction is not required.

The source code for the implementation of OV^{2}SLAM that we plan to use as reference for the challenge is listed in Section 5.3.

2.2 Head Pose Estimation

Before being rendered on the AR HUD, images usually require some forms of correction to accommodate for the real-time position of the driver's viewpoint. Such corrections are usually applied by relying on the output of a eye tracking/gaze estimation or head pose estimation function. For that, there are many factors affecting the choice of the specific algorithm used and its complexity. As anticipated at the beginning of this section, the complexity of these solutions depends on some high-level design parameters of the HUD, like the virtual image distance – longer distances require more complex display hardware, but at the same time can alleviate the complexity of the eye tracking function and allow for simpler methods to be used. In this case study, to keep the problem tractable, we assume that a long virtual image distance is indeed achievable by the HUD and that a head pose estimation method is adequate to solve the issue of determining the driver's viewpoint.

The implementation of the head pose estimation function selected for this case study is Hopenet, which is based on a convolutional neural network (CNN) approach [16]. Hopenet requires a simple RGB monocular camera as input – we assume that another HD camera is installed inside the vehicle, directed towards the driving position, providing input frames via Automotive Ethernet to the SoC at the same target rate of the external cameras (30-60 FPS range).

As CNNs are amenable to hardware acceleration, either through a GPU or through a more dedicated Neural Processing Unit (NPU) or machine learning accelerator, the task should be mapped to one of those compute units, based on the specific platform selected by the challenge participants in their evaluation (Section 4).

In terms of real-time requirements, the head pose estimation task is considered a non-real-time critical, but high-priority, task, as highlighted in Figure 2. The accuracy of the head pose estimation is important, but a small number of frames can be dropped without affecting the general functionality of the application.

There are different versions of Hopenet publicly available, based on different input machine learning frameworks and resolutions/datasets. A reference implementation for Hopenet that we recommend is listed in Section 5.3.

2.3 Aggressor Workloads

While the software tasks required to implement the functionality of the AR HUD (summarized in Figure 2) constitute the main workload for the target system, as part of the challenge we will consider introducing other *aggressor workloads*, competing for the shared hardware resources, that will run alongside the main workload.

Table 1 Parameters characterizing synthetic software tasks, for which reference implementations might not be available (e.g., proprietary/closed-source code bases). Modifications or additions to this list will be agreed with the challenge participants.

Parameter	Units
Compute intensity	FLOPS, IntOPS
Memory access bandwidth	MBps
Input memory buffer capacity	MB
Output memory buffer capacity	MB

In the context of the AR HUD use case, workloads implementing some other functionalities that are typically found in a digital cockpit may be co-located on the same SoC. Figure 3 shows some examples of such workloads. As part of the challenge, we will select one or more of these workloads to be included in the selected use case as aggressor workloads w.r.t. the main AR-HUD workload.

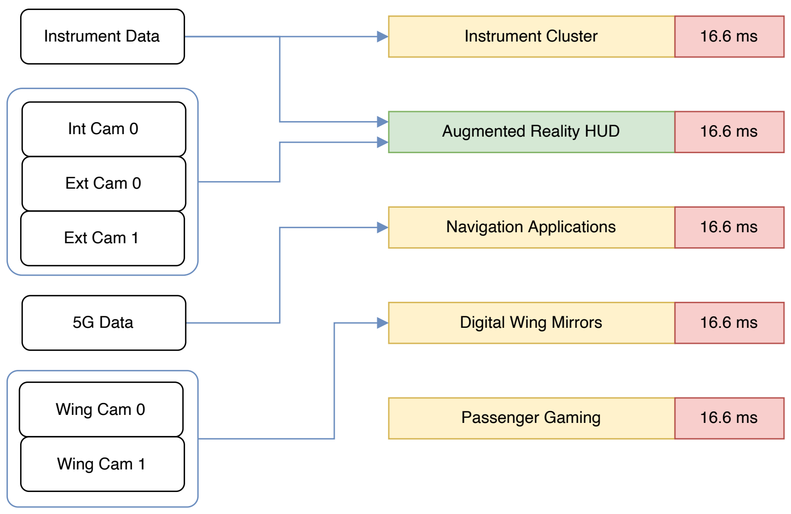


Figure 3 Example of co-located workloads and their deadlines (in red) for a modern digital cockpit.

2.4 Characterization of Tasks

In order to perform the activities described in Section 3, a characterization of the tasks of the considered case study will be required. For those tasks where a public software implementation is available, the characterization can be done by running/simulating the tasks on the selected evaluation platform and by either directly collecting the required statistics or by offline analysis of profiling information gathered during execution (see Section 4 for a description of the suggested evaluation platforms). This is the case, in particular, for the tasks belonging to the SLAM and Head Pose Estimation functions. The characteristics of the remaining tasks, including the aggressor ones, will be shared in the initial phase of the challenge. Different options will then be available on how to model the execution of such synthetic tasks on the evaluation platforms, that are partly discussed in Section 4.

Table 1 reports some key parameters that will be provided for each synthetic task.

2.5 Hardware Target

The target system for the considered case study is a platform comprising a cluster of high-performance Arm A-profile CPU cores, connected to a typical multi-level cache hierarchy configuration. The system will also feature a GPU and optionally an NPU.

The specific details of the hardware target will be shared in the initial phase of the challenge with the participants, and will take into account the specific evaluation platform chosen (Section 4).

3 Challenge Activities

The main tasks of the challenge are:

1. **Analysis of performance bounds:** Perform the response time analysis and worst-case execution time analysis following any methodology deemed suitable to the use case and platform considered. For this step, the challenge participants can choose to assume the absence of shared resources and other observable interference effects. The input for this first step is a description of the software tasks and their dependencies, commonly expressed as direct acyclic graphs (DAGs).
2. **Optimizations:** Perform one or more of the following optimization activities:
   a. Data-flow analysis: Given the challenge use case decomposed into one or more software task DAGs, analyze its data-flows, resource usage and compute requirements, and work on one or more optimizations as follows.
   b. Scheduling: Design one or more scheduling policies that can achieve better system utilization and data sharing between tasks.
   c. Resource mapping: Efficient mapping of tasks to the various hardware components (processing nodes and resources) in the platform, in order to maximise efficiency and minimize contention.
   d. Shared resource interference monitoring and performance isolation: Propose and/or implement shared resources monitoring strategies and design hardware and/or software techniques for shared resource contention mitigation.

3.1 Related Work

This section provides a short survey of real-time analysis techniques proposed in literature, in order to establish a common terminology and provide additional background to the challenge activities described at the top of this section.

3.1.1 Analysis of Performance Bounds

If shared resources in a system are required to provide a deterministic level of service, as defined in Section 1, then the performance of workloads executing on such a system can be found using various known methods, including static analysis, by measurement under worst-case conditions, or by means of formal tools. A static analysis attempts to estimate WCET without actually running the full software on the system. This approach can quickly estimate WCET, however requires the software to be profiled carefully and the traffic simulated on the system in order to come up with an accurate estimate [3, 12]. Another way of estimating performance bounds is by empirical means, i.e., by actually measuring it while running the mixed-criticality tasks on the system [20, 4]. This analysis requires due diligence during run-time for co-locating tasks that result in the highest amount of interference during their execution. We encourage the participants to make use of any tools they see best fit for this challenge and/or they feel most comfortable with. We also encourage the participants to explore more than one technique or even a hybrid approach and to make recommendations based on the achieved results.

3.1.2 Data-flow Analysis

The data-flow model is simple and assumes that a workload can be broken into a series of tasks performed by or with the support of identifiable resources, that co-operate to achieve the higher-level objectives of such a workload in a non-trivial way. Workloads might be dependent on external events or inputs. When this applies, the mapping of tasks to specific resources might need to be determined dynamically at run-time. For tasks that can be executed in parallel (no immediate dependency between the tasks), assigning priorities is not a trivial task. Literature has demonstrated that a classic approach of assigning higher priorities based on the criticality levels and using them during scheduling decisions may result in poor system utilization [1, 19]. Novel run-time robustness mechanisms implemented in the OS/hypervisor layers that support graceful degradation of non-critical tasks partly addresses the system utilization problem [2, 1].

3.1.3 Analysis of Resource Sharing Policies

Shared resources contribute to the execution of a workload by providing one or more services to it. Some form of arbitration, either implicit or explicit, will regulate how users of resources are granted access to those, either in time or space (as in space of the resource) or both. If a resource is shared in time, for it to provide performance isolation, it must hold true that the service it provides to one of its users must be bounded without requiring knowledge of previously serviced users in time. If a resource is shared concurrently (in space), for it to provide performance isolation, it must hold true that the service it provides to one of its users must be bounded without requiring knowledge of concurrently serviced users.

Several techniques have been proposed in literature, e.g., restricting cache line evictions, cache coloring and partitioning, and also regulating memory traffic generated by a particular task in order to reduce the risk of interference between multiple workloads [21, 11, 8]. More recently, dedicated hardware support was added using FPGAs in order to strictly isolate the cores and avoid contention on shared resources such as last-level caches [6]. The focus on literature so far has been heavily around shared memory and bandwidth contention. We envisage that in future systems the problem will start surfacing around domain specific accelerators, that are usually shared across different tasks. This new class of accelerators comes with a different set of constraints due to the limited support for virtualization and preemption of tasks.

3.1.4 Monitoring of Shared Resource Interference

Monitoring shared resources provides insights into what causes contention on shared system resources and enables to implement contention mitigation strategies in the system. Upon determining the activities that make up a workload, we ask the participants how to implement or leverage a monitoring infrastructure that can enable observation of the shared resources involved in the computation of the system workloads. In detail, a monitor infrastructure should allow to determine the utilization of shared resources by their users to enable the setup and enforcement of service level agreements.

4 Evaluation Platforms

While the challenge activities can be addressed with different approaches and tools, we propose two evaluation platforms to be considered by the participants: either a virtual platform or a physical hardware development kit. It will be up to the participants to assess their preferred evaluation strategy for any of the activities. We will provide a comprehensive degree of support for the virtual platform approach, including a software starter kit to serve as baseline for the challenge, in order to streamline platform bring-up. Support for the hardware development kit will instead be provided on a best-effort basis.

Section 5 describes in more details what will be supplied to the participants over the course of the challenge.

Some details of the proposed platforms are reported below:

- Virtual platform: gem5 system-level simulator and AMBA Adaptive Traffic Profiles (ATP)
  - Link (gem5): `https://www.gem5.org/`
  - Link (ATP): `https://github.com/ARM-software/ATP-Engine`
  - Link (gem5 and AMBA ATP): `https://community.arm.com/arm-community-blogs/b/soc-design-and-simulation-blog/posts/amba-atp-engine-3-1-programmable-traffic-generation/`
- Hardware development kit: Jetson Xavier NX
  - Link: `https://www.nvidia.com/en-gb/autonomous-machines/embedded-systems/jetson-xavier-nx/`

As the selected use case will require compute intensive tasks, we propose the adoption of domain-specific accelerators to accelerate parts of it, as suggested in Figure 2. Ideally, we would like to enable modelling the full behaviour of the accelerators on both evaluation platforms. However, depending on the complexity of that approach, we will consider alternatives based on abstracting away the behaviour of the accelerator and other hardware components using traffic generators or approximate/synthetic models.

4.1 Virtual Platform

4.1.1 gem5

gem5 [9] is a modular event-driven simulation platform for computer architecture research, encompassing system-level architecture as well as processor micro-architecture modelling. It is widely used in both academia and industry for rapid early prototyping and/or design space exploration, and it has shown to be an effective tool for providing insights into the impact of system-level interactions when running complex workloads.

Its comprehensive model library (memories, IO devices, etc.) and the architectural support of Armv8-A features (up to Armv8.5-A) allows it to run unmodified complex workloads like Android and boot OSes from UEFI firmware implementations like TFA + u-boot / edkII [7].

Different CPU models, providing different degrees of abstractions and modelling fidelity, are provided in gem5, including two simple single-cycle-per-instruction models (*AtomicCPU*, *TimingCPU*), an in-order pipelined model (*MinorCPU*), and an out-of-order model (*O3CPU*). A memory system can be flexibly built out of caches and crossbars or through the Ruby framework, which provides even more flexible memory system modelling.

gem5 is conceptually a Python library written in C++: the simulated platform is configured in Python (configuration stage), but the instantiated Python models have a matching C++ implementation that gets executed at a later stage (execution stage), once

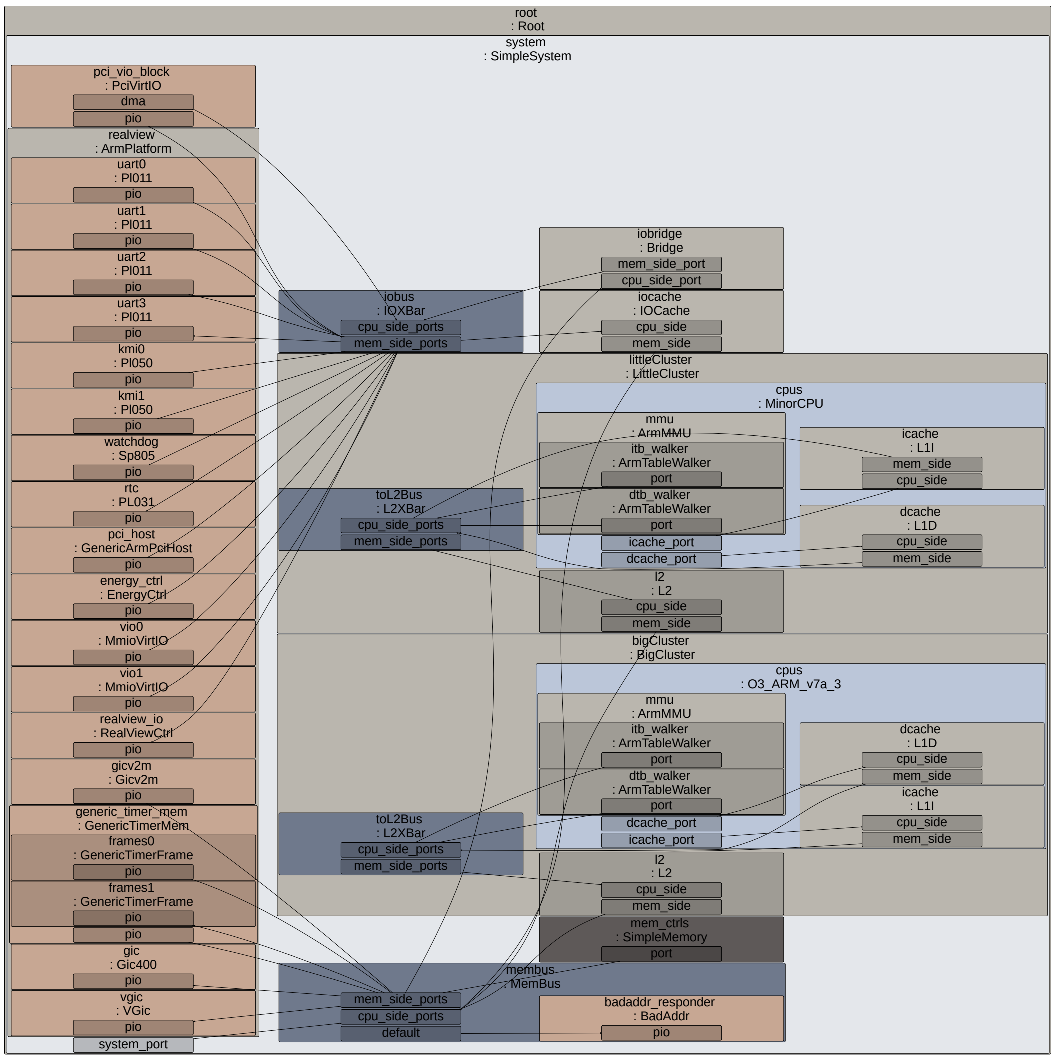
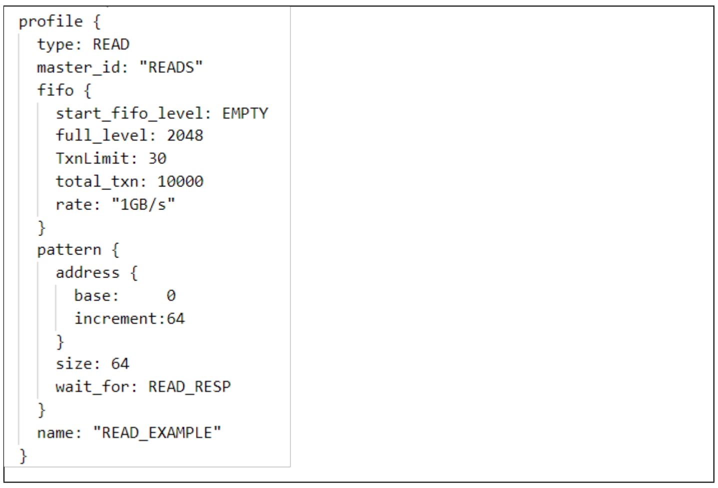


Figure 4 An auto-generated diagram of a system modelled with gem5, including various processing elements, memories and devices.

the simulation is started. This allows to get the best of both worlds: the agility for prototyping and configuring a virtual system in Python, and the execution speed of C++ compiled code, which is crucial to reduce the simulation time for complex systems. Some example configuration platforms are provided within the repository itself (see scripts in `configs/example/arm/`). Those are meant to be starting points for building more complex configurations, and computer architects are expected to extend or adapt them to closely match the system under study.

4.1.2 AMBA Adaptive Traffic Profiles

The AMBA Adaptive Traffic Profiles (ATP) is a definition of the transaction characteristics of an hardware interface. It includes information on the types of transactions and the timing characteristics of those transactions. Traffic profiles can be used during system simulation to represent the behavior of a component. The simulation uses a traffic profile definition to



```
profile {
  type: READ
  master_id: "READS"
  fifo {
    start_fifo_level: EMPTY
    full_level: 2048
    TxnLimit: 30
    total_txn: 10000
    rate: "1GB/s"
  }
  pattern {
    address {
      base:     0
      increment:64
    }
    size: 64
    wait_for: READ_RESP
  }
  name: "READ_EXAMPLE"
}
```

Figure 5 An example of a simple `.atp` file

determine when a particular transaction should be issued, injecting synthetic traffic into the system under study. The AMBA ATP Engine is the open source implementation of the ATP specification. Its backbone is a lightweight FIFO model which injects transactions according to the provided traffic profile (specified through an `.atp` file).

The APT Engine is plugged to gem5 (hosted solution) through an in-tree adaptor (see Figure 6) and this requires to build the ATP Engine as a gem5 loadable module (please follow the ATP Engine `README.md` guide). It is otherwise possible to build gem5 and ATP together with the `meta-atp` layer (link: `https://git.yoctoproject.org/meta-arm/tree/meta-atp/README.md`) from the `meta-arm` repository for Yocto.

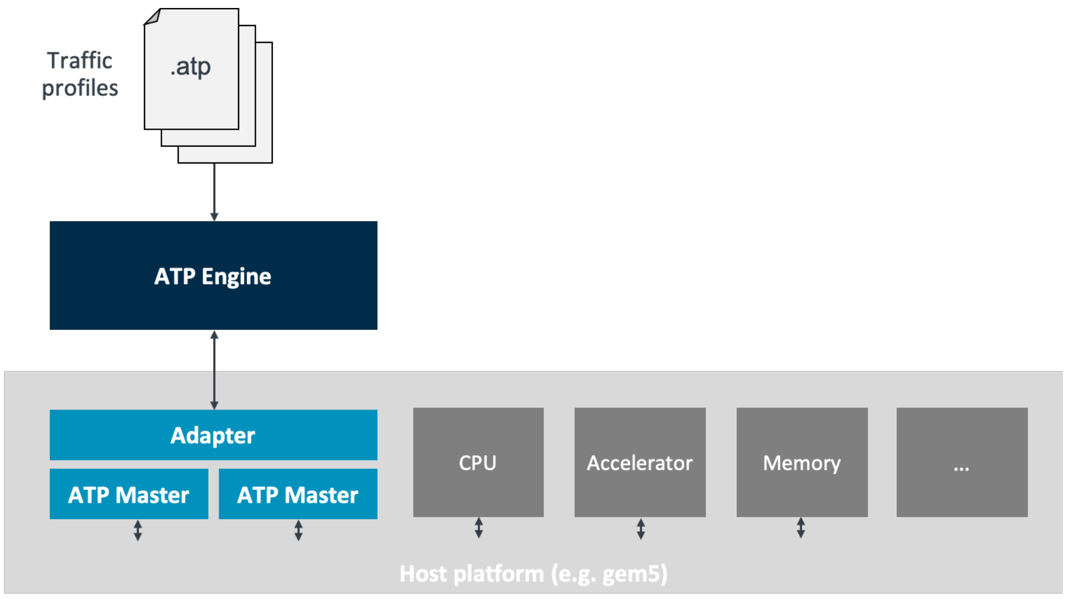


Figure 6 How to model heterogeneous systems by connecting the ATP engine to gem5

These items should be enough to build an evaluation platform using traffic profiles for the computing elements of the system. We won't provide functional NPU/GPU models; if challenge participants will be willing to add functional models of such accelerators to the starter kit simulation platform, prior work on integrating gem5 with approximated models [17, 15] can be used to facilitate the platform bring-up.

4.2 Hardware Development Kit

The hardware platform suggested as alternative to the virtual platform is the Jetson Xavier NX, whose SoC incorporates a 6-core NVIDIA Carmel Armv8.2 64-bit CPU, 384-core NVIDIA Volta GPU with 48 Tensor Cores, and two NVIDIA Deep Learning Accelerator (NVDLA) engines. The latter processing elements could be used to accelerate selected software tasks as specified in Figure 2.

5 Resources

This section provides a list of the resources, including tools, input data sets and profiling data, that will be provided for the challenge, and also the recommended open source implementations of the main software modules of the application in the considered case study.

5.1 Virtual Platform Starter Kit

For the virtual platform solution, we will supply:

- *gem5 starter kit*, including a specific system configuration (detailing platform composition).
- *AMBA ATP profiles*: traffic profiles for all tasks.

5.2 Hardware Development Starter Kit

For the hardware development kit, we will supply:

- *Software tasks source code*: source code for most of the CPU tasks; synthetic "busy-cycle" kernels for the remaining ones (e.g., aggressor tasks).

5.3 Recommended Open Source Software Implementations

- OV^{2}SLAM:
  Implementation of OV^{2}SLAM available on GitHub at the following URL: `https://github.com/ov2slam/ov2slam`. This particular implementation targets CPUs, it leverages multi-threading, and it is written in portable modern C++; it has a few dependencies on widely available libraries and middle-ware (e.g., ROS [18]), which are described at the same URL.
- Hopenet:
  Hopenet is available on GitHub at the following URL: `https://github.com/natanielruiz/deep-head-pose`. For this case study, we would recommend starting from Hopenet-lite, which is a lightweight version of Hopenet based on the simpler ShuffleNet V2 [10] network. The source code for Hopenet-lite is available on GitHub at the following URL: `https://github.com/OverEuro/deep-head-pose-lite`.

6 Conclusions

This document has described a high-performance real-time case study based on an augmented reality head-up display application from the automotive market. This application is a motivating example for the industrial challenge that will be kicked off at the ECRTS 2022 conference. An initial set of questions to prospective challenge participants has been presented, together with initial directions on how to carry out the activities on an Arm-based evaluation platform. Based on the experience from the past editions of the industrial challenge, we expect the definition of the challenge itself to evolve, based on further refinements of the use case and on feedback from the participants. The landing web page for the challenge (`https://www.ecrts.org/arm-industrial-challenge/`) will provide the latest information and it will be used to share deliverables with the real-time research community, including tools, input sets, profiling data, and pointers to reference software implementations. The `#industrial-challenge` Discord channel will be used to address questions from the challenge participants and the research community. We encourage everyone to reach out through the Discord channel or via email directly to the authors for clarifications, feedback and comments.

References

1 Sanjoy Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Haohan Li, Alberto Marchetti-Spaccamela, Nicole Megow, and Leen Stougie. Scheduling real-time mixed-criticality jobs. *IEEE Transactions on Computers*, 61(8):1140–1152, 2012. doi:10.1109/TC.2011.142.

2 S.K. Baruah, A. Burns, and R.I. Davis. Response-time analysis for mixed criticality systems. In *2011 IEEE 32nd Real-Time Systems Symposium*, pages 34–43, 2011. doi:10.1109/RTSS.2011.12.

3 Sudipta Chattopadhyay, Chong Lee Kee, Abhik Roychoudhury, Timon Kelter, Peter Marwedel, and Heiko Falk. A unified wcet analysis framework for multi-core platforms. In *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, pages 99–108, 2012. doi:10.1109/RTAS.2012.26.

4 Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J. Cazorla. Measurement-based probabilistic timing analysis for multi-path programs. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 91–101, 2012. doi:10.1109/ECRTS.2012.31.

5 Maxime Ferrera, Alexandre Eudes, Julien Moras, Martial Sanfourche, and Guy Le Besnerais. Ov^{2}slam : A fully online and versatile visual SLAM for real-time applications. *CoRR*, abs/2102.04060, 2021. arXiv:2102.04060.

6 Giovani Gracioli, Rohan Tabish, Renato Mancuso, reza mirosanlou, Rodolfo Pellizzoni, and Marco Caccamo. Designing mixed criticality applications on modern heterogeneous mpsoc platforms. In *2019 ECRTS*, May 2019.

7 Adrian Herrera. Running trusted firmware-a on gem5. https://community.arm.com/arm-research/b/articles/posts/running-trusted-firmware-a-on-gem5, June 2020.

8 Hyoseung Kim, Arvind Kandhalu, and Ragunathan Rajkumar. A coordinated approach for practical os-level cache management in multi-core real-time systems. In *2013 25th Euromicro Conference on Real-Time Systems*, pages 80–89, 2013. doi:10.1109/ECRTS.2013.19.

9 Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Srikant Bharadwaj, Gabe Black, Gedare Bloom, Bobby R. Bruce, Daniel Rodrigues Carvalho, Jerónimo Castrillón, Lizhong Chen, Nicolas Derumigny, Stephan Diestelhorst, Wendy Elsasser, Marjan Fariborz, Amin Farmahini Farahani, Pouya Fotouhi, Ryan Gambord, Jayneel Gandhi, Dibakar Gope, Thomas Grass, Bagus Hanindhito, Andreas Hansson, Swapnil Haria, Austin Harris, Timothy Hayes, Adrian Herrera, Matthew Horsnell, Syed Ali Raza Jafri, Radhika Jagtap, Hanhwi Jang, Reiley

Jeyapaul, Timothy M. Jones, Matthias Jung, Subash Kannoth, Hamidreza Khaleghzadeh, Yuetsu Kodama, Tushar Krishna, Tommaso Marinelli, Christian Menard, Andrea Mondelli, Tiago Mück, Omar Naji, Krishnendra Nathella, Hoa Nguyen, Nikos Nikoleris, Lena E. Olson, Marc S. Orr, Binh Pham, Pablo Prieto, Trivikram Reddy, Alec Roelke, Mahyar Samani, Andreas Sandberg, Javier Setoain, Boris Shingarov, Matthew D. Sinclair, Tuan Ta, Rahul Thakur, Giacomo Travaglini, Michael Upton, Nilay Vaish, Ilias Vougioukas, Zhengrong Wang, Norbert Wehn, Christian Weis, David A. Wood, Hongil Yoon, and Éder F. Zulian. The gem5 simulator: Version 20.0+. *CoRR*, abs/2007.03152, 2020. `arXiv:2007.03152`.

10 Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018. `doi:10.48550/ARXIV.1807.11164`.

11 Renato Mancuso, Roman Dudko, Emiliano Betti, Marco Cesati, Marco Caccamo, and Rodolfo Pellizzoni. Real-time cache management framework for multi-core architectures. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 45–54, 2013. `doi:10.1109/RTAS.2013.6531078`.

12 Jan Nowotsch, Michael Paulitsch, Daniel Bühler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *2014 26th Euromicro Conference on Real-Time Systems*, pages 109–118, 2014. `doi:10.1109/ECRTS.2014.20`.

13 Panasonic automotive brings expansive, artificial intelligence-enhanced situational awareness to the driver experience with augmented reality head-up display. `https://na.panasonic.com/us/news/panasonic-automotive-brings-expansive-artificial-intelligence-enhanced-situational-awareness-driver`, January 2021.

14 Michael Paulitsch, Oscar Medina Duarte, Hassen Karray, Kevin Mueller, Daniel Muench, and Jan Nowotsch. Mixed-criticality embedded systems – a balance ensuring partitioning and performance. In *2015 Euromicro Conference on Digital System Design*, pages 453–461, 2015. `doi:10.1109/DSD.2015.100`.

15 Samuel Rogers, Joshua Slycord, Mohammadreza Baharani, and Hamed Tabkhi. gem5-salam: A system architecture for llvm-based accelerator modeling. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 471–482, 2020. `doi:10.1109/MICRO50266.2020.00047`.

16 Nataniel Ruiz, Eunji Chong, and James M. Rehg. Fine-grained head pose estimation without keypoints. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

17 Yakun Sophia Shao, Sam Likun Xi, Vijayalakshmi Srinivasan, Gu-Yeon Wei, and David Brooks. Co-designing accelerators and soc interfaces using gem5-aladdin. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12, 2016. `doi:10.1109/MICRO.2016.7783751`.

18 Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL: `https://www.ros.org`.

19 Sebastian Tobuschat, Moritz Neukirchner, Leonardo Ecco, and Rolf Ernst. Workload-aware shaping of shared resource accesses in mixed-criticality systems. In *2014 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, 2014. `doi:10.1145/2656075.2656105`.

20 Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem—overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), May 2008. `doi:10.1145/1347375.1347389`.

21 Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. Memguard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 55–64, 2013. `doi:10.1109/RTAS.2013.6531079`.