

OBSERVATION OF REAL-TIME DEPENDABLE SYSTEMS

ADCSS2023

André Pedro | Research Scientist
andre.pedro@vortex-colab.com

COLLABORATIVE LABORATORY IN
CYBERPHYSICAL SYSTEMS AND CYBERSECURITY

www.vortex-colab.com

TALK OUTLINE

1

INTRODUCTION

CONTEXT & MOTIVATION
CHALLENGES
CONCEPT

2

REFERENCE ARCHITECTURE

AS USUAL
WITH OBSERVATION

3

CONTINUOUS OBSERVATION

BY SOFTWARE
HW-ASSISTED
HYBRID

CONCLUSIONS

1 CONTEXT & MOTIVATION

1 CONTEXT & MOTIVATION

On Microservices

Microservices software architecture is being adopted in general-purpose dependable systems to deal with an exponential increase in software complexity.

However, adoption of cloud-native/edge-native techniques in **safety-critical** and **real-time systems** presents several challenges that need to be overcome, such as constrained resources, isolation, freedom from interference, among others.

Microservices

Orchestration

Container
Runtime

Kernel

Hypervisor

Hardware

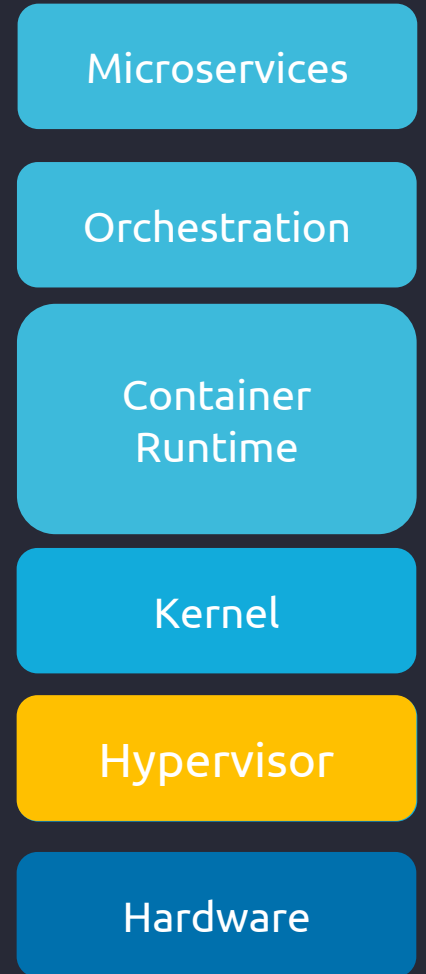
1 CONTEXT & MOTIVATION

On Virtual Machines

Static-Partitioning Hypervisor software is being adopted in real-time systems to deal with the strong isolation of an exponential increase in software components and mixed real-time applications with non-real-time ones.

Using a static-partitioning hypervisor can improve the reliability and safety of real-time systems, while also allowing for greater flexibility and control over the system's resources.

However, It is hard to configure and ensure the given resources are working according to the systems' requirements.



1 CONTEXT & MOTIVATION

On Monitoring

Runtime Monitors are well known as form of watchdogs and are being adopted to act as a safety net, ideally non-intrusive.

- Runtime Monitoring enables continuous observation of real-time dependable systems. The continuous observation of safety-critical embedded systems, in a non-intrusive way, needs co-design of trace features for new SoCs.
- Runtime verification of spatio-temporal properties can also be used for hardware resources.

Watchdogs are transversal

Microservices

Orchestration

Container
Runtime

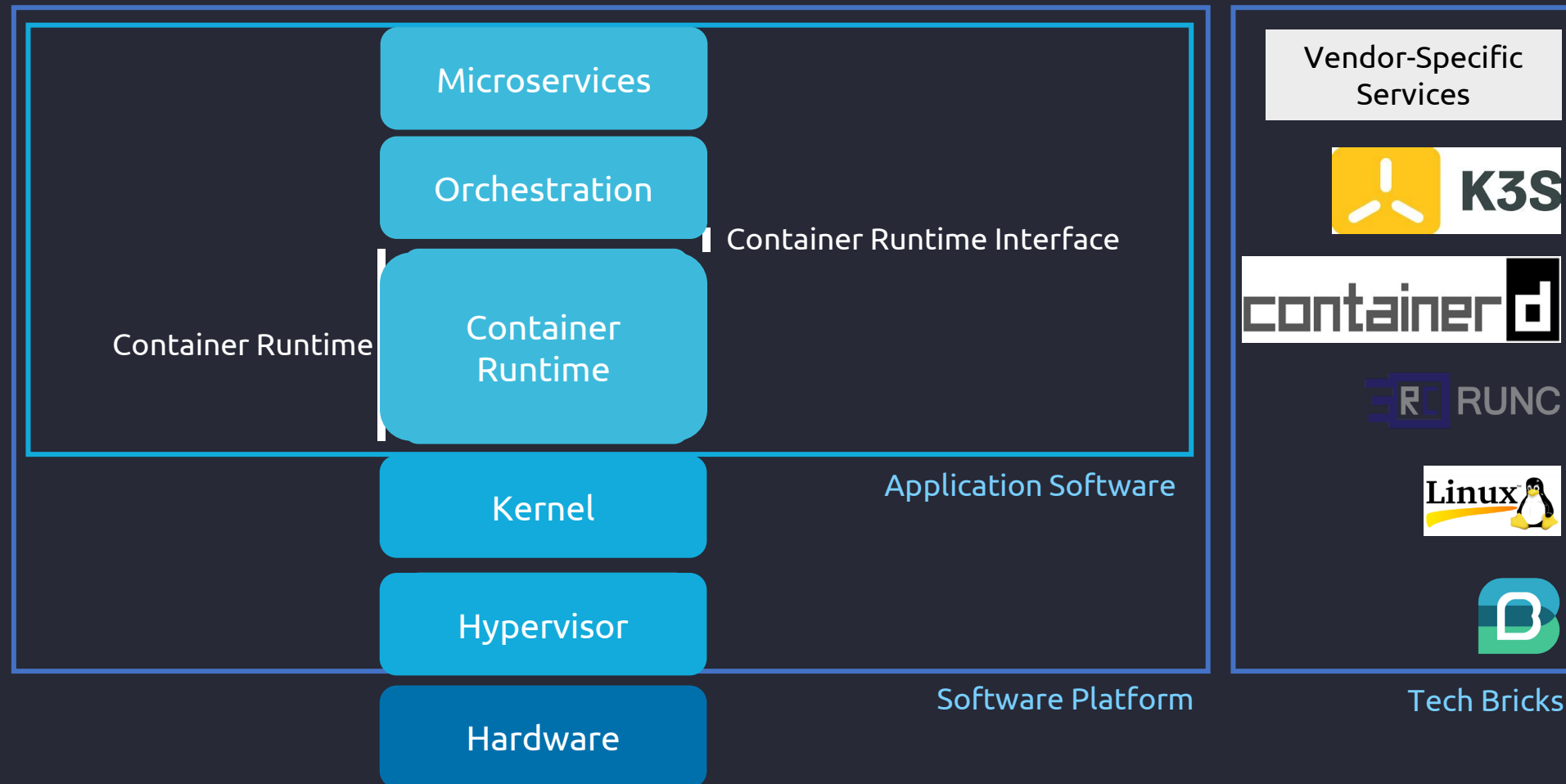
Kernel

Hypervisor

Hardware

1 CONTEXT & MOTIVATION

Example of a Complex Software Stack



Note: The icons and logos used on this slide are for illustrative purposes only.

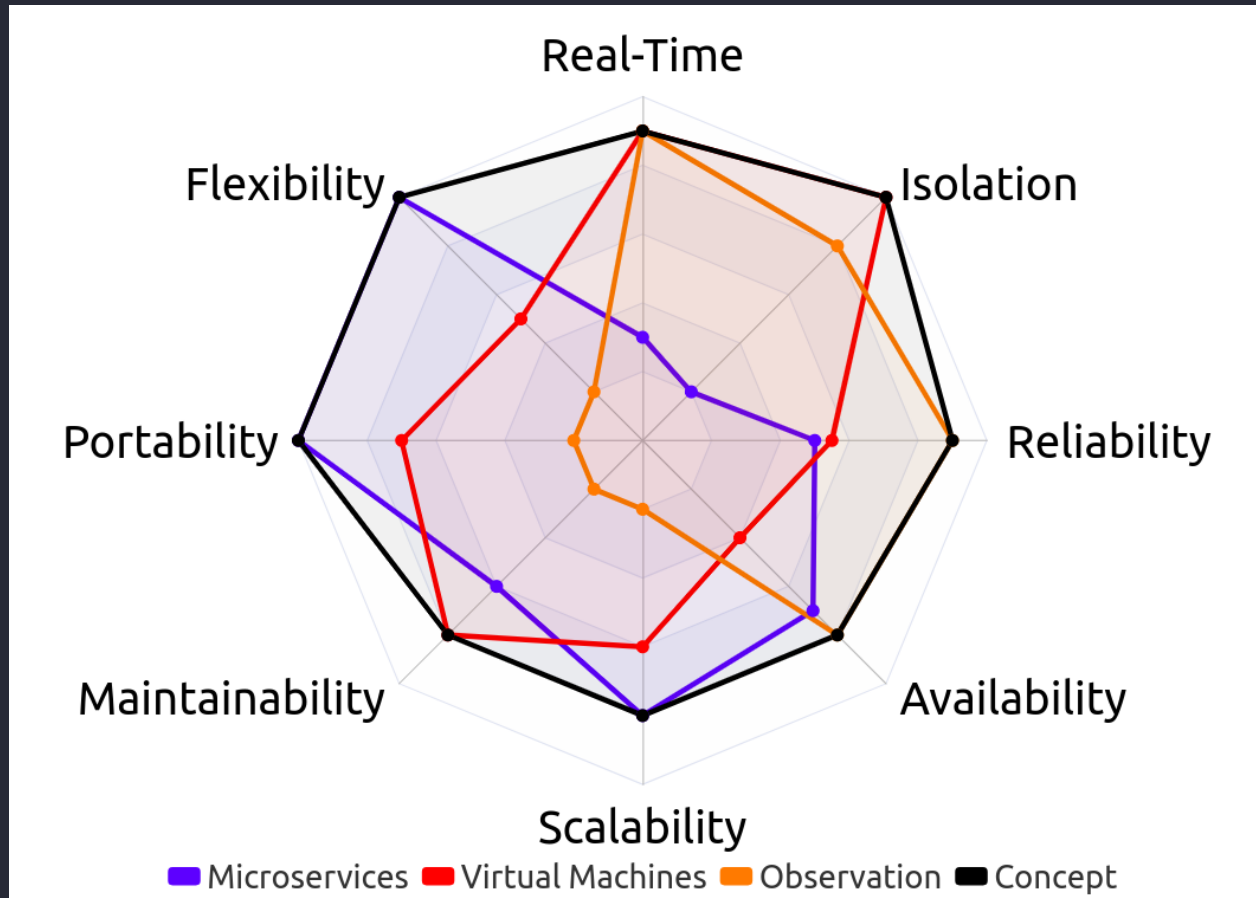
1 CHALLENGES

On Dependable Systems (within our context)

- A concept that clashes with the traditional Conservative Development Cycles
 - Is it possible to show that Correct-by-Design is no longer feasible? Too Flexible
 - Optimize Data Serialization/Deserialization Performance (Latency)
- Microservices**
- Flexibility on Resource Allocation and Control
 - Hard to Debug and Configure (error-prone)
 - Describe the Reference Workload of SW-HW Co-Design
- Virtual Machines**
- Design and Implement Efficient and Correct Monitors
 - Determine the optimal level of resource usage for monitors to avoid mimicking implementations while still effectively observing the system
 - Secure Monitoring and Execution (monitor is the preferred spot for an attacker)
- Runtime Monitoring**

1 CONCEPT

Continuous Observation of Microservices and Virtual Machines



Orchestration, Containerization and Static-Partitioning with Continuous Observation as a Mixed-Criticality Platform with High Reliability.

Note: The polar plot is for illustrative purposes only.

1 CONCEPT

Tradeoffs

PROS

- Microservices allows the system to scale and evolve (add new features, update old features, include new and more reliable and robust features)
- VMM allows the system to run as it is on a different silicon and ideally allows the decoupling of software and hardware
- RV enables More Reliable Software on CoTS Platforms with FPGA-based accelerators
- Monitoring at development and deployment time with CI/CD (Agile Practices; Tracking)
- Provides increased level of safety and security to Edge devices in Space
- Software Segregation is done at different levels. It can help with certification.

CONS

- Neophobia
- Microservices introduces communication latency but they do not result in any execution overhead
- VMs introduces latency and execution overhead on execution units
- VMM Boot time Increase
- It can be argued that excessive flexibility goes against the principle of being correct-by-design
- Complex than Monolithic

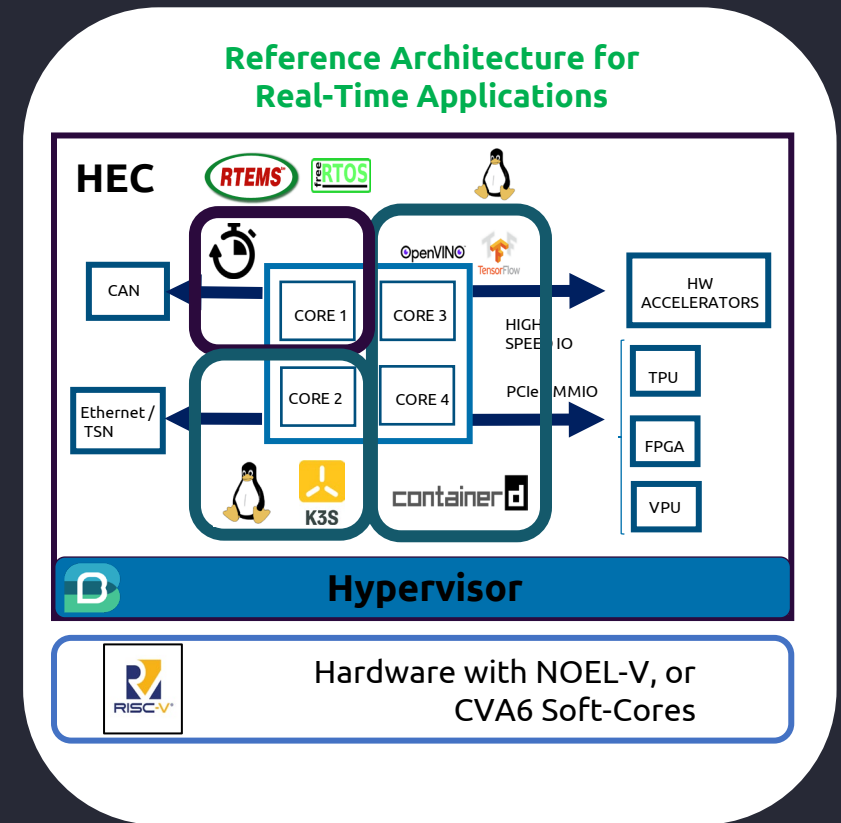
2 REFERENCE ARCHITECTURE

REFERENCE ARCHITECTURE

Real-time Applications with a Virtual Machine Monitor (VMM) or Hypervisor

- VMs guarantee isolation between applications;
- Containers run the microservices in the different VMs;
- Lightweight orchestrator coordinates the microservices running on different VMs;
- A Strong Static-Partitioning Hypervisor allows the mixing of critical microservices with normal microservices and with real-time applications.
- VM runs AI Inference

Note: The icons and logos used on this slide are for illustrative purposes only.

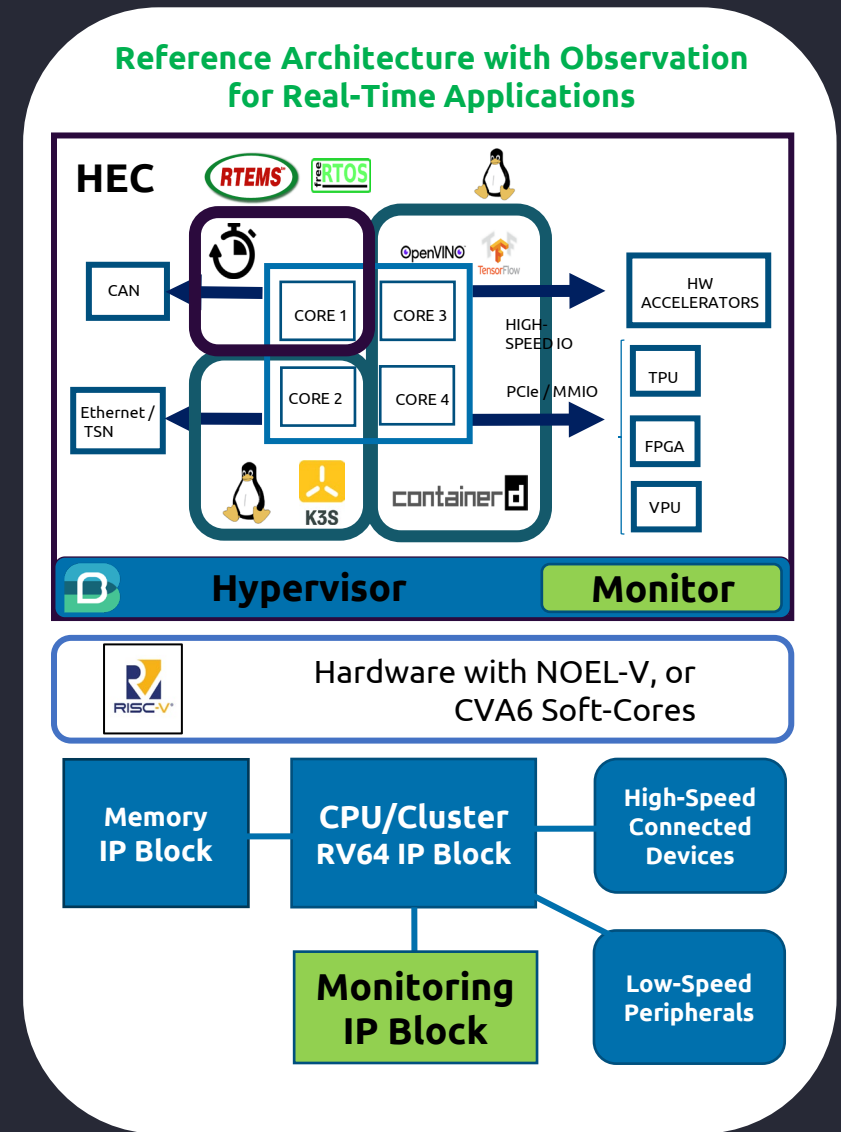


ENABLING SAFETY ON HIGH-END EMBEDDED PLATFORMS

- The future Space **Edge** devices requires an in-depth approach to enable flexibility in resource allocation without discarding guarantees of time, space and energy efficiency. **Observation is key here.**
- Runtime verification can witness these **time, space and energy constraints** (e.g., safety and performance requirements) through continuous observation of the system.
- Continuous observation is a promising feature to achieve **high reliability via non-intrusive monitoring** on a running system.

Monitoring IP Block Add-on for observation of Embedded Applications within a platform

A high-end central computer (HEC) running a strong static-partitioning hypervisor with **high assurance** by continuous observation



3 CONTINUOUS OBSERVATION

VORTEX OBSERVATION SUITE

Enables our Reference Architecture on Modern SoCs

VORTEX Suite enhances the dependability of Embedded Systems by employing automatically generated correct-by-construction monitors to constantly monitor their behaviours. This allows life-critical applications to coexist with non-critical applications, ensuring utmost reliability.

VORTEX Suite has the capability to utilize tracing mechanisms found in modern SoCs, or it can opt for FPGA-assisted tracing and monitoring.

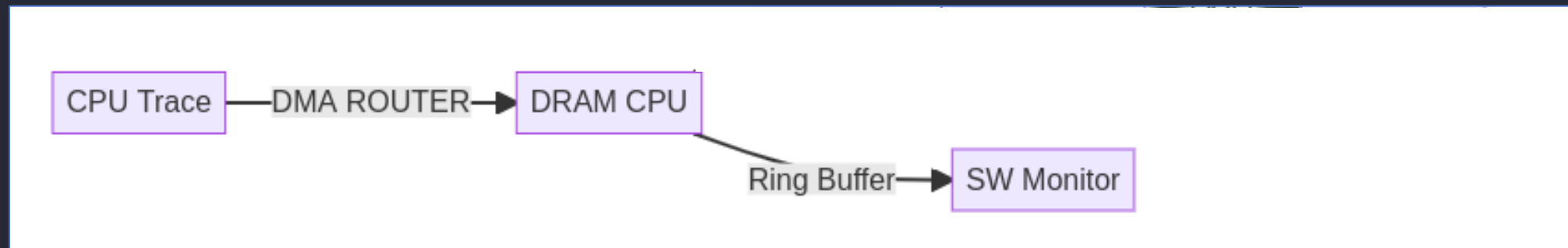
VORTEX Suite provides support for various types of monitors, including:

1. Pure software monitors that are implemented entirely in software.
2. Pure software monitors that run on a dedicated (deterministic) processor core in isolation.
3. Pure hardware monitors that are automatically synthesized from High-Level Specifications.
4. A hybrid approach combining elements of the above methods (WiP).

VORTEX OBSERVATION SUITE

Use Case 1: Pure software monitors that are implemented entirely in software

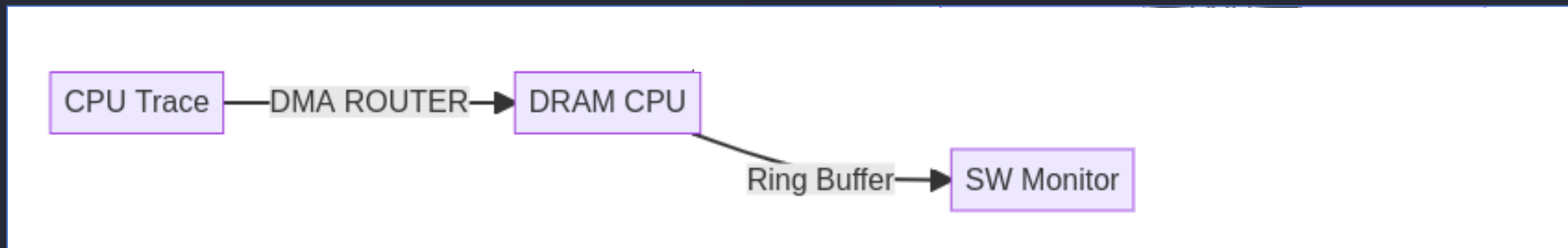
- Formalize the requirement(s) in the specification language
- Generate software monitors based on the formalization
- Locate a memory mapping region that is exclusive for readers (do not perform any writing operations) and another region exclusively for writers (do not perform any reading operations)
- Provide this memory mapping to the OS or Generate Hypervisor Configuration with that memory map



VORTEX OBSERVATION SUITE

Use Case 2: Pure software monitors that run on a dedicated (deterministic) processor core in isolation

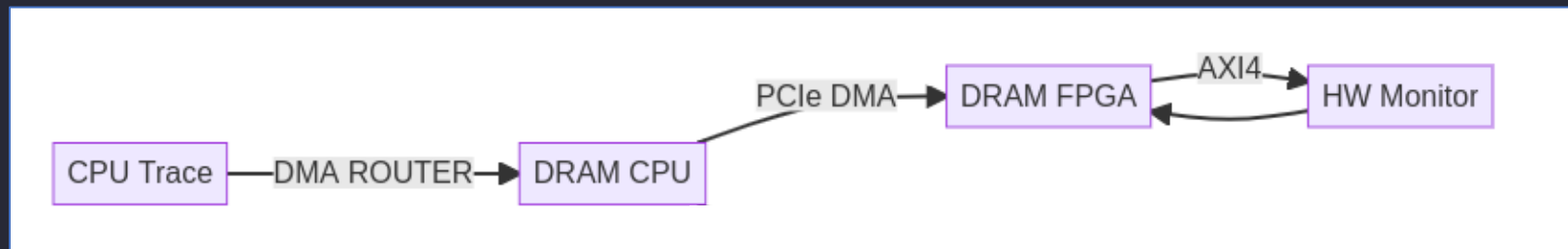
- Formalize the requirement(s) in the specification language
- Generate software monitors based on the formalization
- Locate a memory mapping region that is exclusive for readers (do not perform any writing operations) and another region exclusively for writers (do not perform any reading operations)
- Enable deterministic Memory Mechanisms to receive data on the dedicated core



VORTEX OBSERVATION SUITE

Use Case 3: Pure hardware monitors that are automatically synthesized from High-Level Specifications

- Formalize the requirement(s) in the specification language
- Generate source-code monitors based on the formalization
- Locate a memory mapping region that is exclusive for readers (do not perform any writing operations) and another region exclusively for writers (do not perform any reading operations)
- Generate IP Monitor Block with the Library using HLS
- Add IP to the Reference Design (including the memory mapping)
- Communicate with Hardened Cores via DMA (roughly speaking, registers in the Fabric are mapped to the memory region and circular buffers match in a transparent way)



VORTEX OBSERVATION SUITE

3 Commercial off-the-shelf Single Board Computers (CoTS SBCs)

IMX8QM SoC



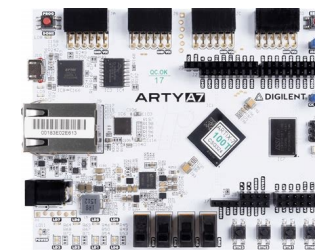
Xilinx Artix-7 FPGA
(tracing offload)



Microchip ICICLE KIT
Polarfire SoC



Arty A7

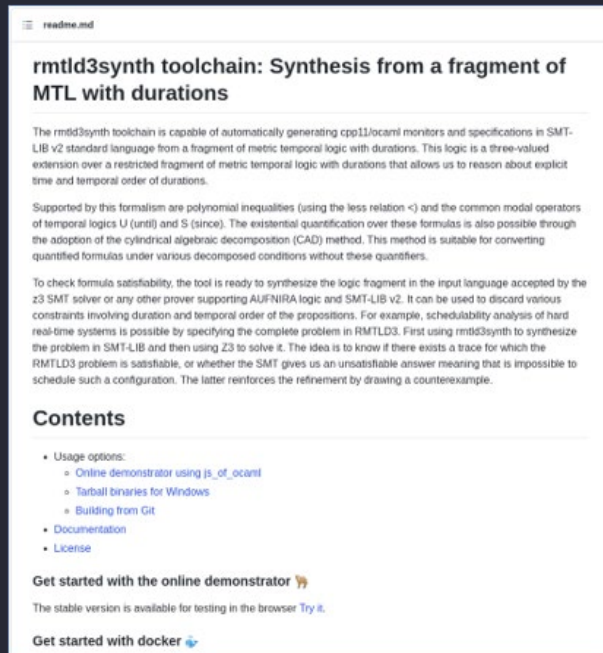


VORTEX OBSERVATION SUITE

What is open/free of use in VORTEX Observation Suite ?

The Toolchain that automatically creates source-code monitors from High-Level Specifications

The Library allows for the integration of real-time monitors as concurrent tasks, either in a lock-free and wait-free manner or through high-level hardware synthesis



rmtld3synth toolchain: Synthesis from a fragment of MTL with durations

The rmtld3synth toolchain is capable of automatically generating cpp11/ocaml monitors and specifications in SMT-LIB v2 standard language from a fragment of metric temporal logic with durations. This logic is a three-valued extension over a restricted fragment of metric temporal logic with durations that allows us to reason about explicit time and temporal order of durations.

Supported by this formalism are polynomial inequalities (using the less relation <) and the common modal operators of temporal logics U (until) and S (since). The existential quantification over these formulas is also possible through the adoption of the cylindrical algebraic decomposition (CAD) method. This method is suitable for converting quantified formulas under various decomposed conditions without these quantifiers.

To check formula satisfiability, the tool is ready to synthesize the logic fragment in the input language accepted by the z3 SMT solver or any other prover supporting AUFNIRA logic and SMT-LIB v2. It can be used to discard various constraints involving duration and temporal order of the propositions. For example, schedulability analysis of hard real-time systems is possible by specifying the complete problem in RMTLD3. First using rmtld3synth to synthesize the problem in SMT-LIB and then using Z3 to solve it. The idea is to know if there exists a trace for which the RMTLD3 problem is satisfiable, or whether the SMT gives us an unsatisfiable answer meaning that it is impossible to schedule such a configuration. The latter reinforces the refinement by drawing a counterexample.

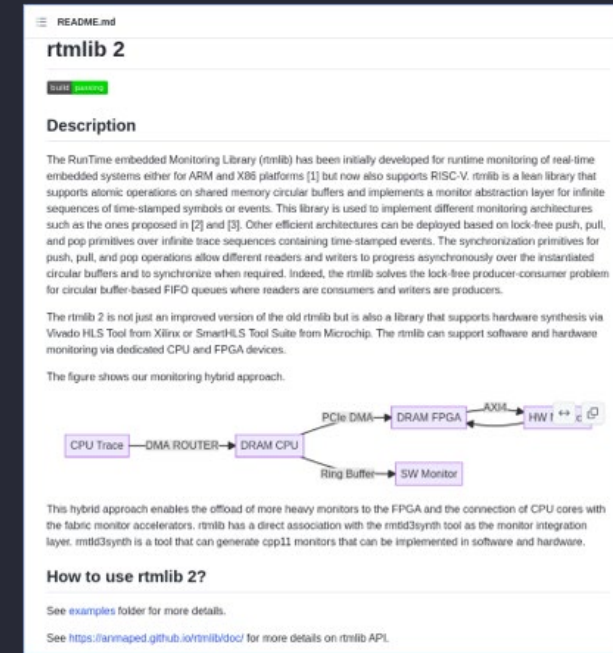
Contents

- Usage options:
 - Online demonstrator using js_of_ocaml
 - Tarball binaries for Windows
 - Building from Git
- Documentation
- License

Get started with the online demonstrator 🚀

The stable version is available for testing in the browser [Try it](#).

Get started with docker 🐳



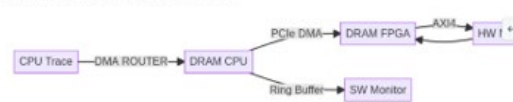
rtmlib 2

Description

The RunTime embedded Monitoring Library (rtmlib) has been initially developed for runtime monitoring of real-time embedded systems either for ARM and X86 platforms [1] but now also supports RISC-V. rtmlib is a lean library that supports atomic operations on shared memory circular buffers and implements a monitor abstraction layer for infinite sequences of time-stamped symbols or events. This library is used to implement different monitoring architectures such as the ones proposed in [2] and [3]. Other efficient architectures can be deployed based on lock-free push, pull, and pop primitives over infinite trace sequences containing time-stamped events. The synchronization primitives for push, pull, and pop operators allow different readers and writers to progress asynchronously over the instantiated circular buffers and to synchronize when required. Indeed, the rtmlib solves the lock-free producer-consumer problem for circular buffer-based FIFO queues where readers are consumers and writers are producers.

The rtmlib 2 is not just an improved version of the old rtmlib but is also a library that supports hardware synthesis via Vivado HLS Tool from Xilinx or SmartHLS Tool Suite from Microchip. The rtmlib can support software and hardware monitoring via dedicated CPU and FPGA devices.

The figure shows our monitoring hybrid approach.



```
graph LR
    CPU[CPU Traces] --> DMA[DMA ROUTER]
    DMA --> DRAM[DRAM CPU]
    DMA --> FPGA[DRAM FPGA]
    DRAM --> SW[SW Monitor]
    FPGA -- AXI4 --> HW[HW]
    HW --> SW
```

This hybrid approach enables the offload of more heavy monitors to the FPGA and the connection of CPU cores with the fabric monitor accelerators. rtmlib has a direct association with the rmtld3synth tool as the monitor integration layer. rmtld3synth is a tool that can generate cpp11 monitors that can be implemented in software and hardware.

How to use rtmlib 2?

See [examples](#) folder for more details.

See <https://anmaped.github.io/rtmlib/doc/> for more details on rtmlib API.

Including examples of coupling the monitors

Available in <https://github.com/anmaped/rmtld3synth> and <https://github.com/anmaped/rtmlib>.

VORTEX OBSERVATION SUITE

What is closed in VORTEX Observation Suite ?

- The AI-assisted formalization of high-level requirements to be used by our tools
- The Library that couple tailored Hardware Monitors
- The tailoring of the observation of Data and Instructions on NOEL-V Soft-Core (WiP)
- The Demos and Examples for Commercial Platforms
- Our new IP Block (WIP) that accelerates software monitors but also enables (low-level) monitoring of
 1. Bus Contention and Bandwidth (e.g., AMBA),
 2. Quality of service in Network-on-Chips, and
 3. Variety of non-functional behaviours with Data and Instruction level operations for NOEL-V Cores.

CONCLUSIONS

- Our automated approach enables the generation of monitors from high-level specification languages
- These specification languages have the ability to reason about both time and space.
- With this approach, we can observe a vast range of behaviours without manual construction of monitors
- The scalability of our solution ensures that complex monitors can be generated as needed, especially with the increasing demand in AI applications
- The behaviours are transversal to the software and hardware architecture
- The behaviours observed by our monitors span across both software and hardware architecture, ideally ensuring requirements coverage throughout the entire system
- Most importantly, our generated monitors are trustworthy

OBSERVATION OF REAL-TIME DEPENDABLE SYSTEMS

ADCSS2023

André Pedro | Research Scientist
andre.pedro@vortex-colab.com

COLLABORATIVE LABORATORY IN
CYBERPHYSICAL SYSTEMS AND CYBERSECURITY

www.vortex-colab.com

OTHER SLIDES