Enforcing Timeliness and Safety in Mission-Critical Systems

António Casimiro, Inês Gouveia, José Rufino
casim@ciencias.ulisboa.pt
http://www.di.fc.ul.pt/~casim

LaSIGE, Faculdade de Ciências,
Universidade de Lisboa, Portugal

Ciências ULisboa LASIGE

73rd IFIP WG 10.4 Meeting
Goa, India, January 15, 2018

Motivation

- Cyber-physical systems involve complex interactions with the environment and also dealing with uncertainty
  - E.g., autonomous vehicles will be increasingly connected, to their surrounding environment and to each other, thus depending on information received from external sensors
- Ensuring safety despite uncertainties is a hard problem
  - Often addressed by designing the system for the worst possible scenario (**but with implications on performance or cost**)
- We proposed the **KARYON** hybrid system model and architecture to address this problem
  - Separating the system into a complex part and a Safety Kernel
  - The Safety Kernel must be timely and reliable

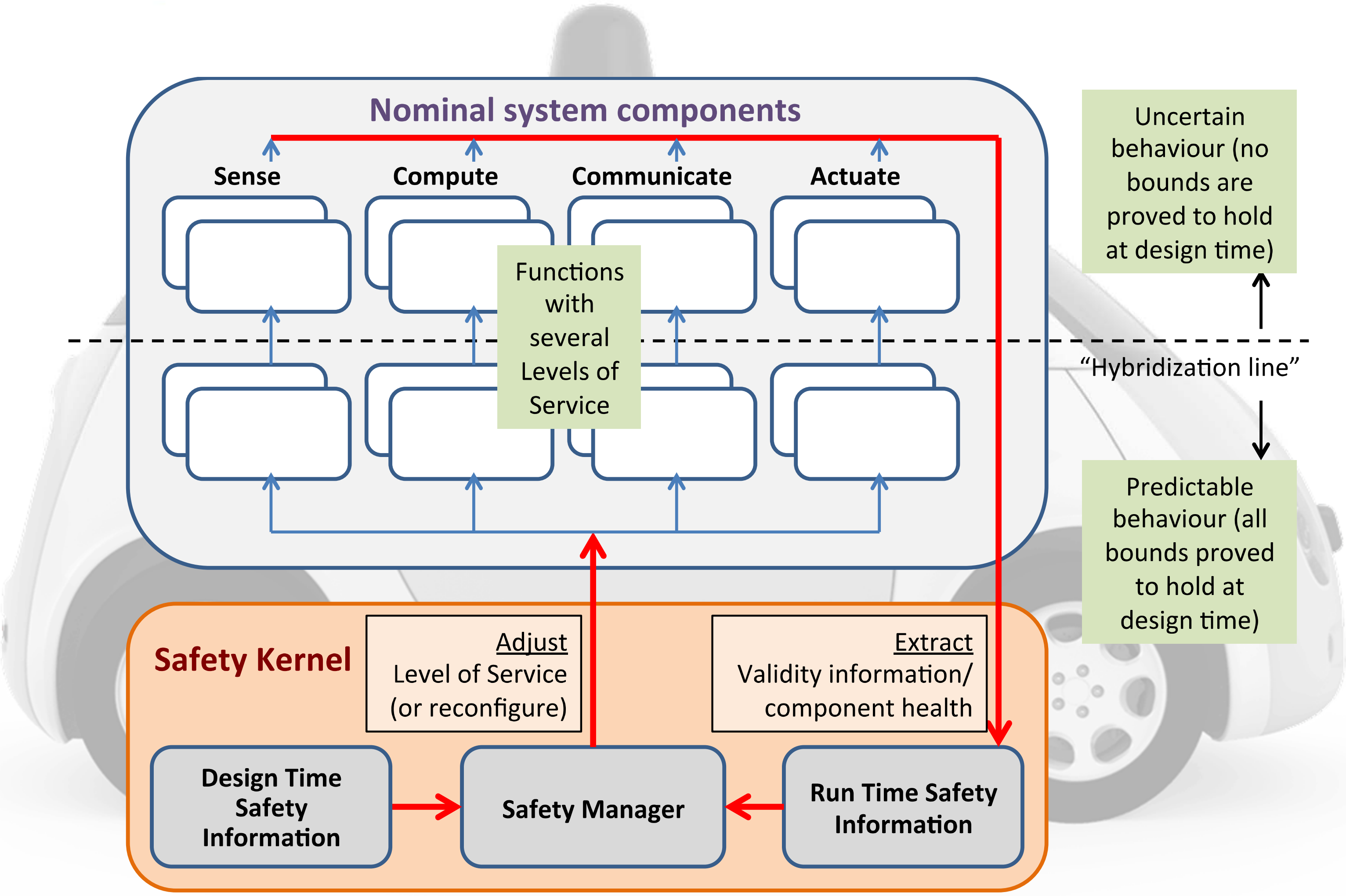
Motivation

- For safety reasons, it is fundamental that the properties of the critical parts of the system (namely the Safety Kernel) are satisfied with a very high probability
- Is there something that might be done if some critical property is violated in runtime? (despite all measures that might have been taken to enforce them)

We propose a hardware-based non-intrusive runtime verification approach to detect possible violations of critical properties

Applying hardware-based non-intrusive runtime verification to the KARYON Safety Kernel

Safety Kernel



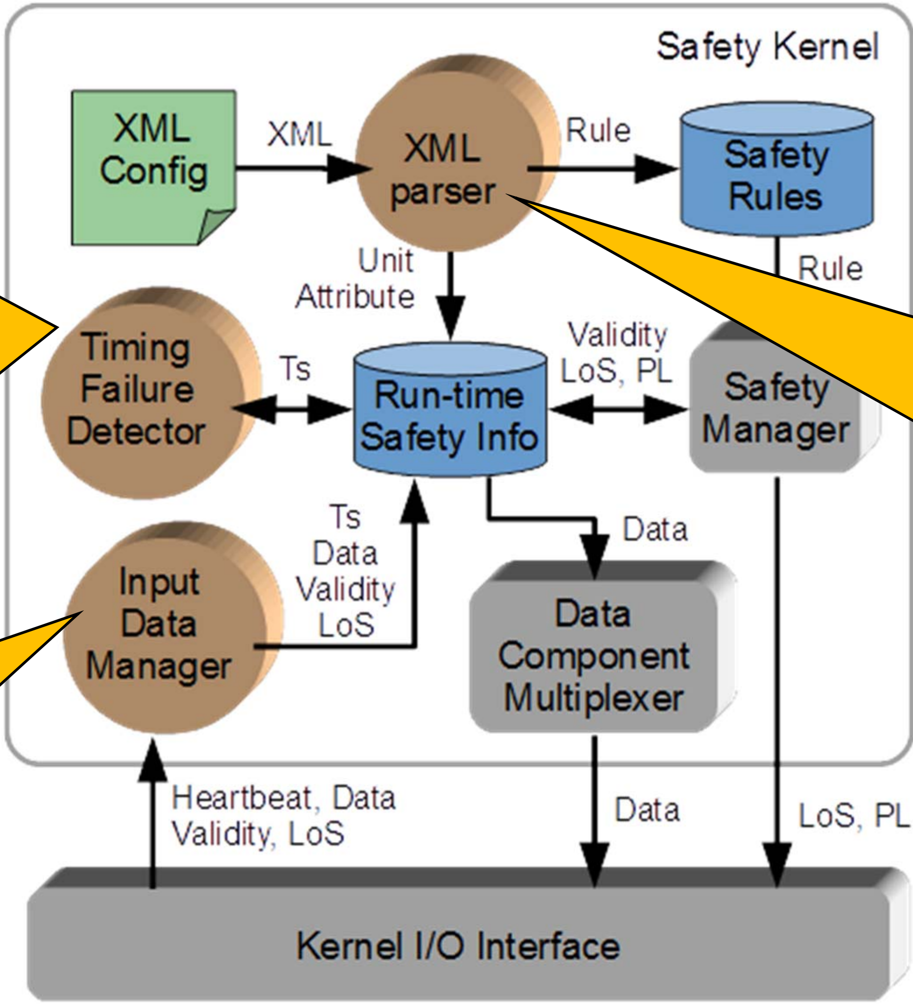
Safety Kernel operation

- The safety kernel continuously collects information on the integrity and timeliness or validity of data in the nominal system, which varies over time
- And adjusts the Level of Service (LoS) of the functions executed by the nominal system (e.g., preventing the use of components whose integrity is not sufficiently high), aiming to operate in the highest possible LoS
- In design time, it is proven that functionality is safe in each of the possible LoS, as long as a set of defined safety rules for each LoS are satisfied
- The Safety Kernel selects the LoS by checking which safety rules are satisfied, given the collected data validity and timeliness information

Safety Kernel architecture

Periodic thread: periodically runs the **TFD** and the **Safety Manager** to evaluate safety rules and determine the acceptable LoS

Listener thread: collects heartbeats (**timeliness** info) and **validity** info



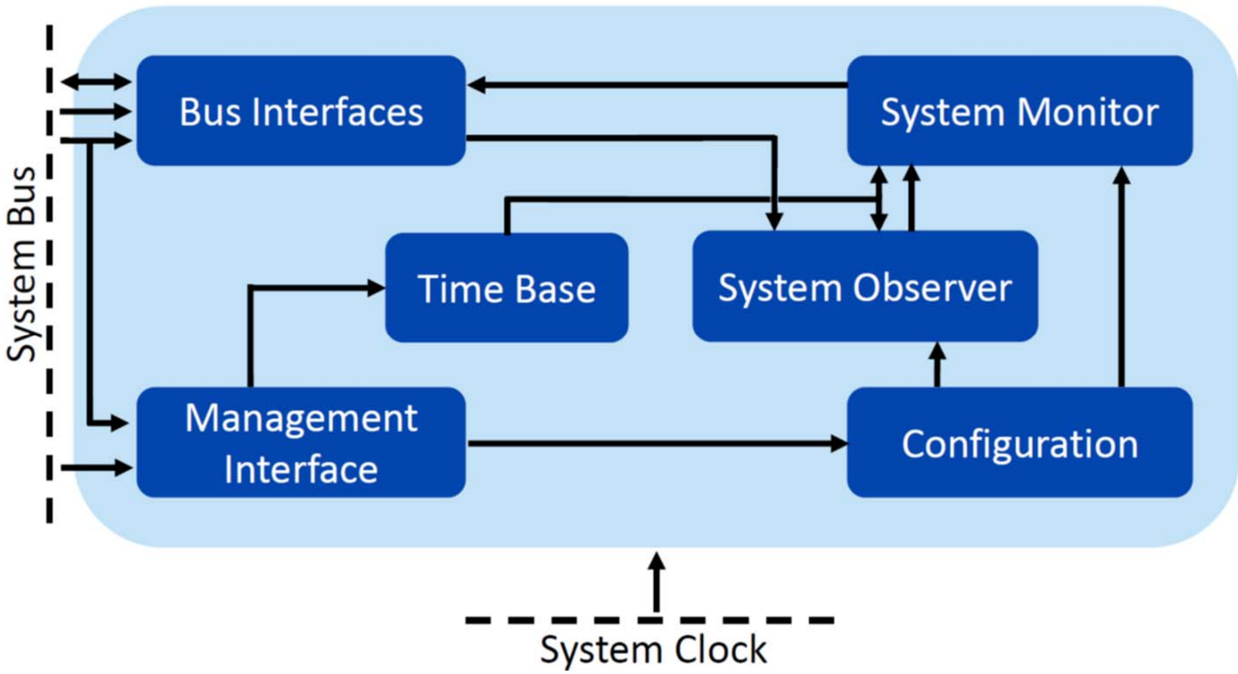
Initialization thread: parses XML file containing safety rules and build structures in repositories

Safety Kernel assumptions

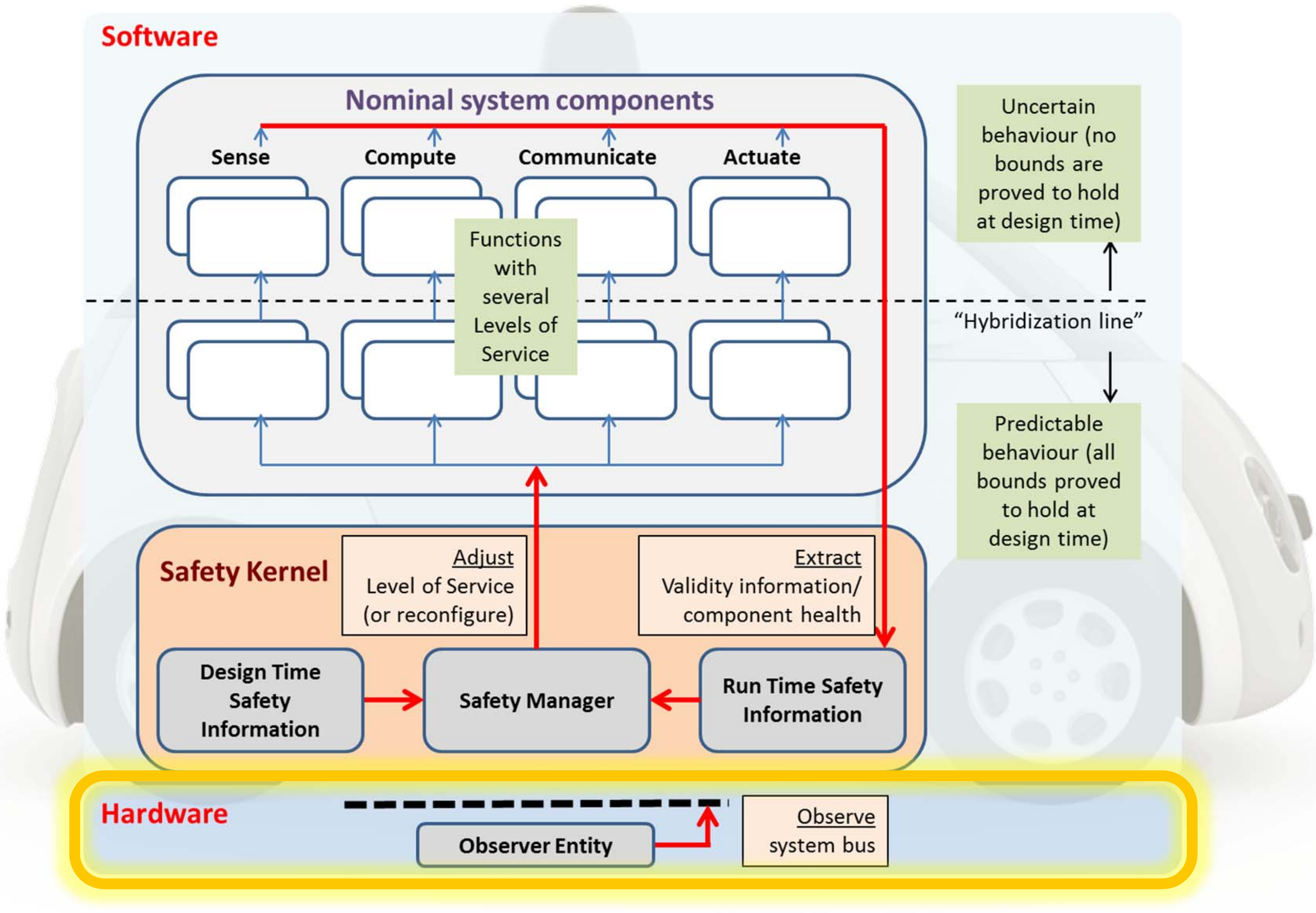
- Bounded input (for listener thread):
  - The **number of received packets** (heartbeats, validity indications) is bounded by $N_{packets}$
  - It is hard to enforce this bound at design time because the nominal system might malfunction and send too many packets to the Safety Kernel
- Bounded execution time (of periodic thread):
  - The **execution time** of each Safety Kernel job is bounded by D_{SK}
  - This bound might be violated only when some fault interferes with the (expectedly predictable) execution time of the Safety Kernel tasks

Non-intrusive runtime monitor

- Runtime verification of assumptions is performed by an **Observer Entity** that may be implemented using versatile FPGA-based platforms



Observer entity & Safety Kernel



Verifying SK assumptions

- Bounded input ($N_{packets}$)
  - Initialize the Observer Entity **counting monitor** with $N_{packets}$ whenever a new instance of SK process starts

How? By configuring the address of first instruction as an event of interest, linking the event to the counting monitor

  - Decrement counter whenever a packet is received

How? By configuring the address of a relevant instruction within the listener thread as an event of interest

  - Detect violation when counter is smaller than zero
  - Call an exception handler (if it exists) to deal with such unforeseen situations
    - E.g., start manoeuvre to stop the car, because a critical component for the vehicle safety is not working properly

Verifying SK assumptions

- Bounded execution time (D_{SK})
  - Initialize the Observer Entity **timeliness monitor** with D_{SK} when new instance of SK process starts

How? Addresses of first and last instructions will be used as events of interest to start/stop the time counter

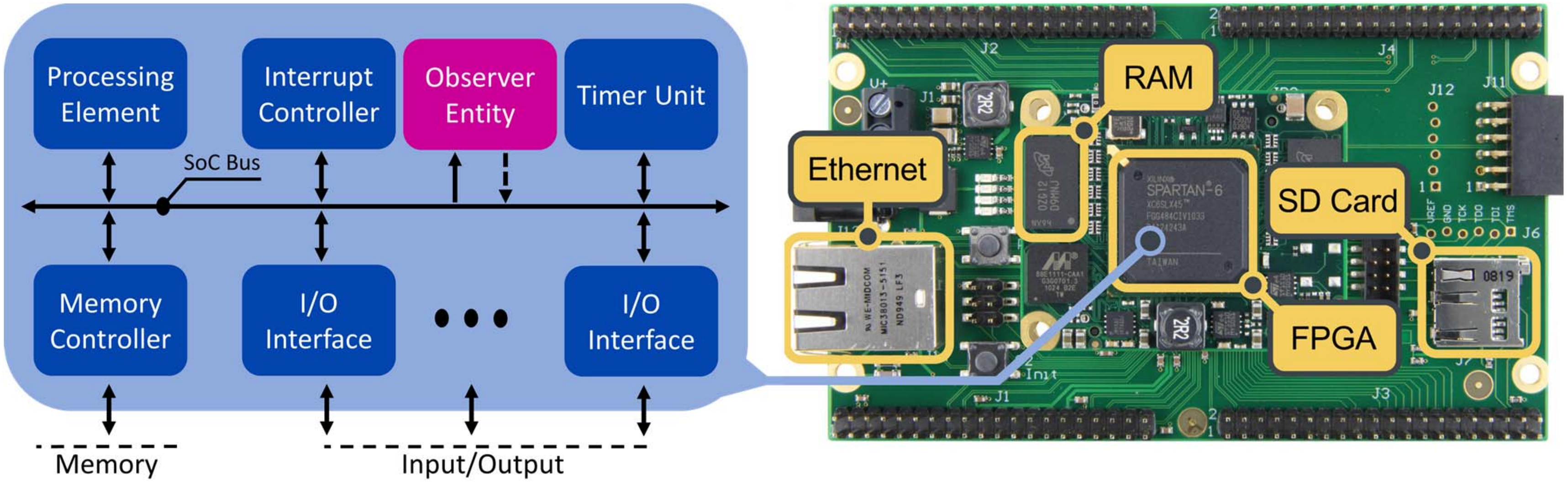
  - Decrement time counter at each system clock tick
  - Detect violation when counter is smaller than zero
  - Stop time counter when the SK process ends

  - Like before, call an exception handling if a violation is detected

Plans for validating the idea

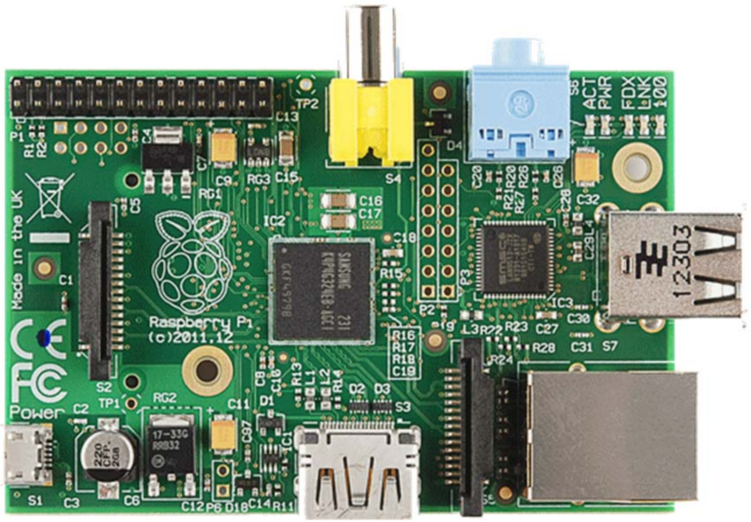
Implementation on soft-processor

- **FPGA-based** development board
- Processing unit: **LEON3** soft-processor (SPARC v8 arch)
- **RTEMS** executing on top
- Support for TSP on RTEMS allows for hybrid system architecture
  - Nominal system may be on separate hardware, connected to the board through some of its interfaces (e.g., Ethernet)
- Available resources are adequate to support the Observer Entity

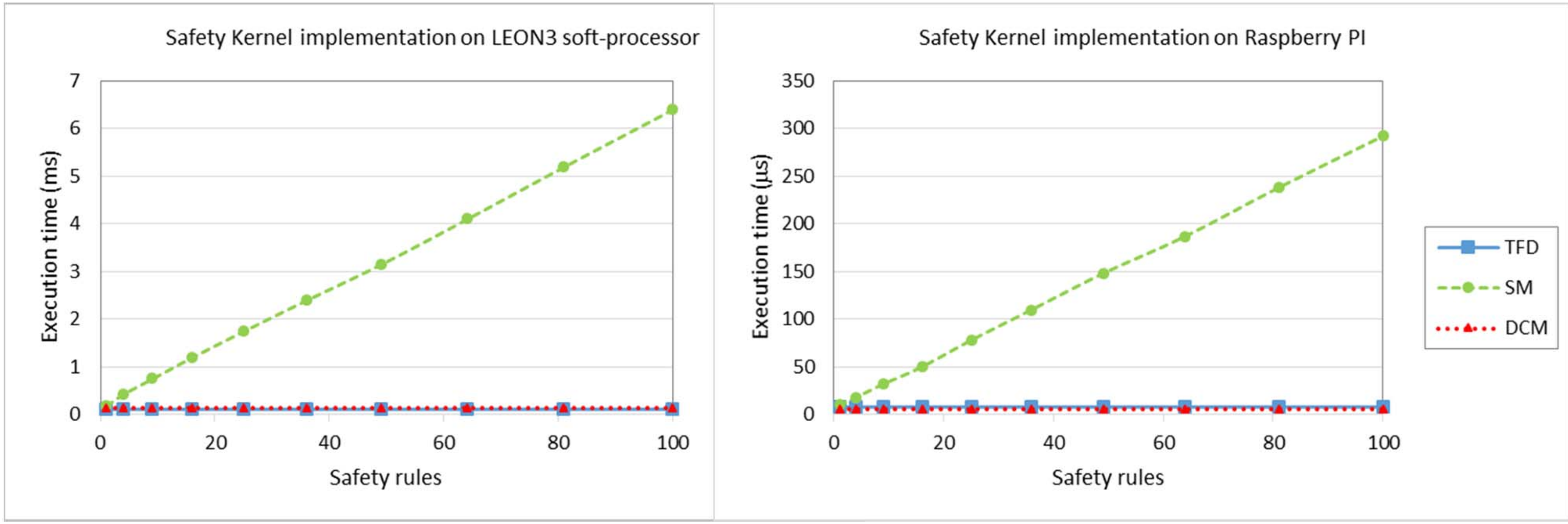


Implementation on Raspberry PI

- **Raspberry PI** Model B Rev 2.0
- **ARM 11** processor (700MHz)
- **Real-Time Linux**
- No support for hybridization nor for non-intrusive runtime verification
- But good to **compare the performance of a soft-core processor (LEON3) with a real core (ARM)** while running the Safety Kernel



Performance comparison



- The execution time is mostly determined by the Safety Manager (SM) component, which processes the safety rules
- **Using a real processor significantly improves the performance** (about 20x in this case)

Conclusions

- Adding non-intrusive runtime verification is **important to detect the violation of design assumptions**, while it can be ignored and has no interference on the system if no violations are detected
- It may significantly contribute to **enhance the overall system dependability**
- The idea is to do a real implementation on a FPGA-enabled system, with a soft-processor, to show feasibility
- Integration on ARM processors requires ARM CoreSight facilities

Thank you for your attention!

To reach me: casim@ciencias.ulisboa.pt
Web page: http://www.di.fc.ul.pt/~casim