SafeX: Open Source Hardware and Software Components for Safety-Critical Systems

Sergi Alcaide[†], Guillem Cabo[†], Francisco Bas[†,‡], Pedro Benedicte[†], Francisco Fuentes[†],
Feng Chang[†], Ilham Lasfar[†], Ramon Canal[‡,†], Jaume Abella[†]
[†]Barcelona Supercomputing Center (BSC)
[‡]Universitat Politècnica de Catalunya (UPC)

***Abstract*—RISC-V Instruction Set Architecture (ISA) emerges as an opportunity to develop open source hardware without being subject to expensive licenses or export restrictions. A plethora of initiatives are nowadays developing systems-on-chip (SoCs) and its components based on RISC-V targeting a wide variety of markets. However, domains with safety requirements, such as avionics, space, and automotive, impose SoCs to include support to meet those requirements.**

This work introduces the SafeX family of components, a set of components providing SoC controllability, observability and safety measures support. These components, developed by the Barcelona Supercomputing Center with permissive open source licenses, are intended to be the basis to make SoCs meet the needs of domains with safety requirements. In particular, the SafeX components developed so far include the SafeSU (multicore statistics unit), the SafeTI (flexible and programmable traffic injector), the SafeDE and SafeSoftDR (hardware and software modules to enforce lockstep execution), and the SafeDM (module to monitor diversity across cores).

***Index Terms*—safety, observability, controllability, MPSoC**

I. Introduction

The RISC-V Instruction Set Architecture (ISA) [12] has become extremely popular with the democratization of hardware design, which makes hardware development – either in the form of IP or actual SoCs – affordable even for small companies and research institutions. In particular, RISC-V ISA becomes a vehicle to develop microprocessors without inheriting ISA licensing costs, as opposed to commercial ISAs, and without relevant export restrictions, hence easing the development and distribution (including commercialization) of RISC-V based products worldwide.

In this context, a plethora of processors and SoCs have emerged, with some of them becoming highly popular. Only at the RISC-V International web portal [12] one can already find more than 100 cores and SoCs, including IPs from SiFive, Codasip, ETH Zurich/U. Bologna, Syntacore, Andes, UC Berkeley, CloudBEAR, and Microchip, to name a few. For instance, some popular cores and SoCs include LowRISC's Ibex core, Ariane and PULPino SoCs by ETH Zurich and the University of Bologna, and a number of cores and SoCs based on the Rocket one by SiFive and UC Berkeley.

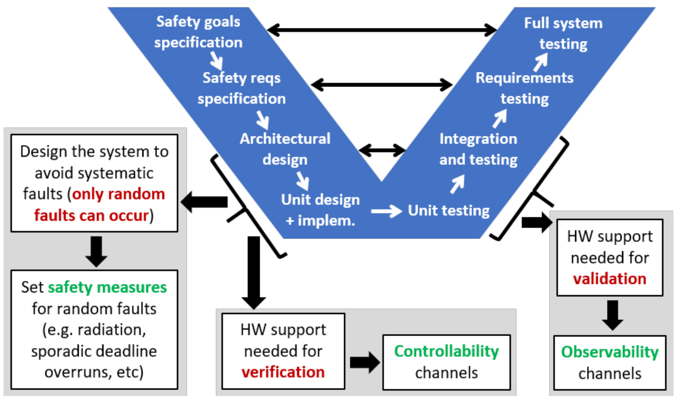
Unfortunately, in general, those cores and SoCs do not offer the support needed for their use in safety-related systems, such as appropriate observability and controllability means, and support to implement safety measures (e.g., Dual Core Lockstep – DCLS). To our knowledge, only two product families in the RISC-V arena provide safety compliance: CAES Gaisler's NOEL-V core and SoC [6] for the space domain, and NSI-TEXE's NS31A and other CPUs [11] for the automotive domain. However, while part of NOEL-V IP is offered as open source, this excludes safety and reliability support, which is only distributed under commercial licenses. Analogously, NSI-TEXE's IP is not open source. Hence, there is a gap in the RISC-V open source domain to provide cores and SoCs meeting the requirements of safety-related systems.

To respond to this situation, the Computer Architecture and Operating Systems interface (CAOS) group at the Barcelona Supercomputing Center (BSC) is developing the *SafeX* components family: a family of RISC-V compliant open source – with permissive licences – hardware and software components intended to enable RISC-V cores and SoCs meet the requirements needed for their adoption in safety-related systems. In particular, the CAOS group has already developed some components including the following:

- **SafeSU** [3], [4]. A multicore interference aware hardware statistics unit providing observability capabilities to identify sources of interference, and controllability means to set interference quotas.
- **SafeTI** [13]. A flexible and programmable hardware traffic injector easing the test of functional and non-functional features of an SoC.
- **SafeDM** [2]. A hardware module measuring the diversity across two cores – typically running a task redundantly – to support safety measures to manage scenarios with lack of diversity.
- **SafeDE** [1]. A hardware module enforcing diversity through time staggering across two cores – typically running a task redundantly – to allow implementing a form of DCLS.
- **SafeSoftDR** [10]. The software-only counterpart of SafeDE to allow implementing a form of DCLS without explicit hardware support.

Some of them are already offered as open source at https://bsccaos.github.io/, whereas the rest will be progressively released as their implementation reaches enough maturity, which is proven through their integration in commercial and highly mature SoC prototypes by CAES Gaisler developed in the framework of H2020 De-RISC [7] and H2020 SELENE projects [14]. Moreover, other components providing watchdog and DCLS capabilities are already in the CAOS roadmap.

The rest of this paper presents some background on the development process of a safety-relevant system, and the foreseen architecture of a safety-relevant microprocessor



Safety goals specification
Safety reqs specification
Architectural design
Unit design + implem.
Unit testing
Integration and testing
Requirements testing
Full system testing
Design the system to avoid systematic faults (only random faults can occur)
Set safety measures for random faults (e.g. radiation, sporadic deadline overruns, etc)
HW support needed for verification
Controllability channels
HW support needed for validation
Observability channels

Fig. 1. V-model of the development process of a safety-relevant system.

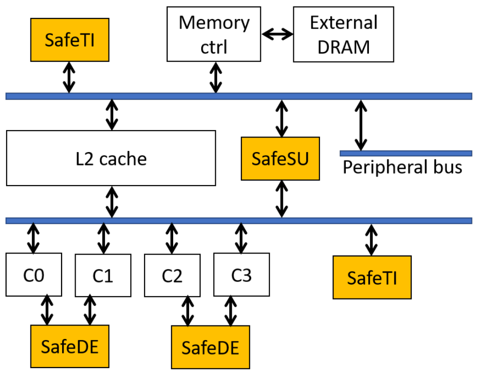


Fig. 2. Example of safety-relevant microprocessor with SafeX hardware modules integrated.

in Section II, the already developed SafeX components in Section III, an illustrative example of use in Section IV, and some conclusions in Section V.

II. Background

A. Development Process of a Safety-Relevant System

Safety-related systems follow a development process dictated by the corresponding domain-specific functional safety standards, such as ISO26262 in automotive, DO178C and DO254 in avionics, EN5012x in railway, etc. Such development process can be abstracted as a V-model, as shown in Figure 1. First, safety goals are specified and safety requirements derived from those goals. The system is architected accordingly mapping safety requirements to components so that the latter jointly fulfill those requirements. When architecting the system, safety measures needed to manage unavoidable random hardware faults must already be included. Then, the different units composing the system architecture are designed and implemented. Verification steps are already conducted at model and implementation level to determine whether the system model and its components meet their requirements by design. On the right side of the V-model, testing (validation) activities start bottom up – from individual components to the fully integrated system – gathering empirical evidence of the adherence to specifications.

In that development process, safety measures must be planned and effectively implemented during the architectural design and implementation. In the same process stages verification occurs, which normally requires controllability means to enforce specific behavior of the components and the system model. Finally, during the testing campaign, observability means become crucial to gather detailed evidence used to assess whether the system behaves according to its specifications with relevant data. Therefore, appropriate support is needed to deploy safety measures, to exercise control of the system, and to achieve sufficient observability.

B. A Safety-Relevant Microprocessor

SafeX components have no specific constraint to be integrated on a wide variety of microprocessor architectures. However, they have already been integrated in an SoC like the one illustrated in Figure 2, which shows the main architecture used in H2020 De-RISC [15] and H2020 SELENE [8].

As shown, the microprocessor includes some cores connected through a bus. Such bus implements the Advanced Microcontroller Bus Architecture (AMBA) and, in particular, the AMBA Advanced High-performance Bus (AHB) architecture. A shared second level (L2) cache is connected to the AMBA AHB bus on one hand, and to an AMBA Advanced eXtensible Interface 4 (AXI4) on the other, where the latter is used to reach DRAM memory and peripherals. Note that, while SafeX components have been integrated in such a microprocessor, they are not strictly limited to such a microprocessor or to its specific communication protocols, although porting SafeX components to a different SoC or communication protocol requires tailoring their interfaces.

The details on how each SafeX component interacts with the different SoC components are provided in next section along with the description of the SafeX components.

III. SafeX Components

This section introduces the SafeX components developed so far providing support to implement safety measures (SafeSU, SafeDM, SafeDE and SafeSoftDR), to provide observability channels (SafeSU), and to provide controllability means (SafeSU and SafeTI).

A. SafeSU Hardware Statistics Unit

The SafeSU is a multi-purpose multicore statistics unit providing means to implement safety measures, as well as observability and controllability channels [3]. In particular, the SafeSU incorporates three main features:

- Cycle Contention Stack (CCS). The CCS [9] is a mechanism which, based on the arbitration signals observed in the bus, determines what bus master (e.g., typically a core) is using the bus – if any, and what master is being delayed by another bus master that may be using the bus. Hence, it allows breaking down the time a task is willing to use the bus into actual utilization and interference created by each other core.
- Maximum-Contention Control Unit (MCCU). The MCCU [5] provides support to set interference quotas across cores so that, upon a quota exhaustion, an interrupt is raised allowing the operating system or the hypervisor to take the corresponding corrective action.

- Request Duration Counters (RDCs). The RDCs [5] provide capabilities to measure the largest request duration observed per request type so that such information can be used, for instance, for Worst-Case Execution Time (WCET) estimation. The RDCs also provide capabilities to pre-program them with a threshold value, and raise an interrupt if the actual request duration observed exceeds the pre-programmed value, hence indicating that an overly long event has been observed.

Apart from all those features, already available in the open source prototype, we are currently in the process of extending the CCS to monitor interference in interfaces where the master ids do not correspond to the cores, as illustrated in Figure 2. For instance, if two cores create interference on each other in the access to DRAM beyond the L2 cache, their master ids are not explicitly visible in the L2-to-DRAM interface since the master id identifies the L2 cache, not the core generating the request. As part of H2020 De-RISC and H2020 SELENE, we are developing two alternative solutions to quantify interference and break it down across cores based on other signals (e.g., L2 cache misses) or on modules propagating core ids.

Note that the SafeSU has been extended recently with reliability features as part of a Failure Mode and Effect Analysis (FMEA) [4] to better adhere to the requirements of safety-relevant systems.

B. *SafeTI Hardware Traffic Injector*

The SafeTI is a flexible and programmable traffic injector [13]. It works attached to a communication interface, as shown in Figure 2, where we show that there may be multiple SafeTI modules integrated in the SoC, either in different interconnects (like in the figure) or in the same interconnect. The publicly available version supports AMBA AHB, but a version also supporting AMBA AXI4 is currently under development.

The SafeTI is programmed with specific commands that allow performing an action repeatedly or a sequence of specific actions. Such pattern with one or multiple actions can be, in turn, performed once, a given number of times, or repeated indefinitely. Actions include read and write operations with pre-defined targets, and with varying amounts of data transmitted, hence modelling transmission bursts. Actions also include delays of pre-defined duration so that read and write transactions occur at specific rates.

The SafeTI can be used for multiple purposes, being the main one generating specific stress patterns to assess the impact of timing interference on tasks running in the cores. Such patterns can be devised to mimic synchronously – and hence in a controlled manner – the traffic patterns that asynchronous components such as accelerators, DMAs, and peripherals could generate. This way, the impact of those traffic patterns can be tested in a controlled manner.

C. *SafeDM Hardware Diversity Monitor*

The SafeDM is a hardware monitor that, by comparing the state of the pipeline of two cores, determines whether they have enough diversity so that a fault that could affect both of them (e.g., a voltage droop), cannot lead both cores to experiencing the same error [2]. Faults leading to identical errors in redundant cores are referred to as *Common Cause Failures (CCFs)* in automotive jargon. The typical solution to mitigate CCFs is using DCLS. However, due to the complexity of today's SoCs, if a task is run redundantly in two cores, it is extremely unlikely to have the same state in both cores since tasks will not easily get synchronized (e.g., due to serialization accessing DRAM). However, while diversity is expected to exist, means are needed to provide such evidence to build the safety concept, and SafeDM provides such evidence.

SafeDM builds on gathering information from the pipeline registers of the cores as a proxy of their current *electrical state* and comparing those values across cores. If they differ, then current is flowing heterogeneously across cores and any disturbance affecting both of them will produce different electrical, and hence also logical, effects. Therefore, by comparing the outcomes of the tasks running in those cores, even if both have erroneous state, the error will be detected because errors will differ.

Note that SafeDM may raise false positives, hence indicating lack of diversity when it exists due to non-monitored activities. However, our evaluations show that false positives are extremely unlikely. In fact, we have observed false positives for very small loops with almost identical behavior across iterations so that, even if staggering exists, may lead to identical sets of instructions and data being processed across both cores.

In terms of operation, SafeDM is a non-intrusive module since it does not interfere with the execution of the tasks of the cores being monitored. SafeDM just snoops specific information through dedicated signals and collects information that can be read at will. So far, SafeDM has been integrated with NOEL-V cores [6], but nothing precludes its integration with other cores.

D. *SafeDE Hardware Diversity Enforcement Module*

The SafeDE works as a light version of DCLS for non-lockstep cores [1]. Tasks need to be run redundantly by software means, and SafeDE is in charge of guaranteeing that the trail core does not catch up with the head core. This way, both cores execute different instructions at any point in time and, upon a fault affecting both cores simultaneously, no CCF can happen.

The SafeDE snoops the instruction counts of both cores and checks whether the difference in number of instructions executed by the head core and the trail core is above a pre-defined threshold. If the difference is below the threshold (typically as many instructions as a core may have in-flight simultaneously), then the trail core is stalled using the appropriate stall signal. If, instead, the difference is above the threshold, then both cores are allowed to progress, resuming the execution of the trail core if it was stalled.

The SafeDE has negligible performance impact since the execution of the task in the trail core is typically delayed by few tens of cycles with respect to that of the task in the head core. Hence, SafeDE provides a very light way to achieve DCLS. Note, however, that the effectiveness of the SafeDE module depends on whether the instruction stream executed

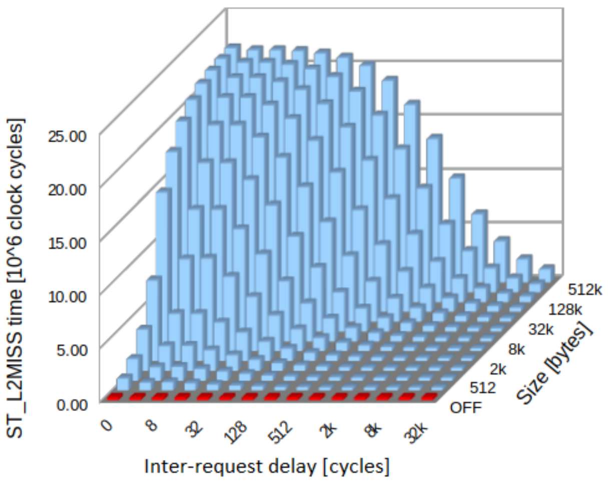


Fig. 3. Experiment example with a benchmark experiencing interference from the SafeTI, and using the SafeSU to measure such interference.

redundantly in both cores is truly identical. Upon a divergence, then the guarantees in terms of execution staggering are lost.

As in the case of SafeDM, SafeDE has been integrated with NOEL-V cores [6], but nothing precludes its integration with other cores.

E. SafeSoftDR Software Diversity Enforcement Library

The SafeSoftDR is the software-only counterpart of SafeDE [10]. In particular, the SafeSoftDR implements a software monitor polling both the head and trail cores, to obtain how many instructions have executed each of them, and compares the difference against a pre-defined threshold. Since the monitoring loop takes much longer when performed at software level, the staggering threshold also needs to be much higher (e.g., 100,000 instructions), which makes staggering be as much as 100μs.

The SafeSoftDR makes execution in the trail core stall and resume building on typical process signals such as SIG_STOP and SIG_CONT in Linux, but analogous signals can be used for other operating systems.

Overall, SafeSoftDR has non-negligible impact in the execution time of the task running in the trail core, which will finish some hundreds of microseconds later than that running in the head core. However, SafeSoftDR does not require any hardware support and can be deployed virtually in any RISC-V multicore.

IV. Illustrative Example

For illustrative purposes, we show the joint use of SafeTI and SafeSU in the context of the SELENE platform [8] in Figure 3. In particular, we show the measured interference observed when running a benchmark performing sustained write memory accesses in one of the cores, whereas the SafeTI injects different types of write traffic. Traffic injected consists of requests with varying size (between 512 bytes and 1MB, see z-axis), and with varying delay between requests (between 0 and 32K cycles, see x-axis). Interference experienced by the benchmark measured with the SafeSU (in millions of cycles, see y-axis) shows how increasing request sizes increase interference, as well as increasing inter-request delays reduce such interference. As shown, the SafeTI effectively and flexibly injects interference, whereas the SafeSU is capable of monitoring it. More complex examples (e.g., with multiple SafeTI modules or adding traffic from other cores) could be produced, and the SafeSU would allow to break such interference down across contenders.

V. Conclusions and Future Work

This work introduces the SafeX family of components for safety-relevant RISC-V microprocessors. Components developed so far include statistics units (SafeSU), traffic injectors (SafeTI), and support for diverse redundancy (SafeDM, SafeDE and SafeSoftDR). Some of those components are already offered with permissive open source licenses, and the rest of them will be distributed analogously in the near future.

By incorporating SafeX components, some key features needed by safety-relevant microprocessors can be achieved, hence closing the gap towards fully open source safety-relevant RISC-V microprocessors.

Part of our future work consists of porting SafeX components to other SoCs, other cores, and other interfaces (e.g., AMBA ACE), as well as developing additional components complementing already developed ones, such as, for instance, watchdogs and true DCLS layers.

Acknowledgements

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 871467. This work has also been partially supported by the Spanish Ministry of Science and Innovation under grant PID2019-107255GB-C21 funded by MCIN/AEI/10.13039/501100011033.

References

[1] F. Bas et al. SafeDE: a flexible diversity enforcement hardware module for light-lockstepping. In *IOLTS*, 2021.
[2] F. Bas et al. SafeDM: a hardware diversity monitor for redundant execution on non-lockstepped cores. In *DATE*, 2022.
[3] G. Cabo et al. SafeSU: an extended statistics unit for multicore timing interference. In *ETS*, 2021.
[4] G. Cabo et al. SafeSU-2: Safe statistics unit for space MPSoCs. In *DATE*, 2022.
[5] J. Cardona et al. Maximum-Contention Control Unit (MCCU): Resource access count and contention time enforcement. In *DATE*, 2019.
[6] Cobham Gaisler. NOEL-V Processor. https://www.gaisler.com/index.php/products/processors/noel-v.
[7] De-RISC Consortium. De-RISC website, 2021. https://www.derisc-project.eu/ (accessed Feb-2021).
[8] C. Hernàndez et al. Selene: Self-monitored dependable platform for high-performance safety-critical systems. In *DSD*, 2020.
[9] J. Jalle et al. Contention-aware performance monitoring counter support for real-time MPSoCs. In *SIES*, 2016.
[10] F. Mazzocchetti et al. SafeSoftDR: a library to enable software-based diverse redundancy for safety-critical tasks. In *FORECAST Workshop (with HiPEAC conference)*, 2022.
[11] NSI-TEXE. NS31A : RISC-V 32bit CPU which supports ISO26262 ASIL D. https://www.nsitexe.com/en/ip-solutions/ns-series/ns31a/.
[12] RISC-V International. RISC-V International website. https://riscv.org/.
[13] O. Sala et al. SafeTI: a hardware traffic injector for mpsoc functional and timing validation. In *IOLTS*, 2021.
[14] SELENE Consortium. SELENE website, 2021. https://www.selene-project.eu/ (accessed Feb-2021).
[15] N.-J. Wessman et al. De-risc: the first risc-v space-grade platform for safety-critical systems. In *SCC*, 2021.