SafeSU-2L: An Advanced Multicore Interference Statistics Unit for a RISC-V Space SoC

Francisco Bas†
Sergi Alcaide†
Guillem Cabo†
Ilham Lasfar†
Feng Chang†
Francisco Fuentes†
Ramon Canal†,‡
Pedro Benedicte†
Juan Carlos Rodriguez†
Jaume Abella†, Senior Member, IEEE

†Barcelona Supercomputing Center (BSC), Spain
‡Universitat Politecnica de Catalunya (UPC), Spain

***Abstract*— Multicores for critical real-time embedded systems (CRTES) may experience interference across tasks running in different cores when accessing shared hardware resources such as shared caches and memory controllers. Precise interference diagnostics (e.g., what task interferes what other task and how much) are key for the optimization and validation of safety-related real-time applications during development, and to diagnose overruns during operation. The SafeSU statistics unit has been proposed recently for that purpose, and proven successful for systems-on-chip (SoCs) where interference can occur at a single centralized locations (e.g., a bus). However, it is unable to monitor interference in multiple-level interconnects, especially if request ownership is not available.**

This paper extends the SafeSU to 2-level interconnects (SafeSU-2L), where interference can occur in the bus connecting the cores with a shared second level cache (L2C), and in the DDR4 memory controller serving L2C misses, with the latter losing track of the actual core issuing each DDR4 request. In particular, the SafeSU-2L monitors some additional signals from the buses and caches to infer what core interferes what other core in any of the shared resources. Moreover, the SafeSU-2L is integrated and tested on a 4-core version of the commercial CAES Gaisler NOEL-XCKU-EX space SoC.

***Index Terms*— Statistics unit, interference, multicore, space, RISC-V.**

Manuscript received XXXXX 00, 0000; revised XXXXX 00, 0000; accepted XXXXX 00, 0000.
The research leading to these results has received funding from the Horizon Europe Programme under the NimbleAI Project (nimbleai.eu), grant agreement num. 101070679. Authors appreciate the support given to the Research Group SSAS (Code: 2021 SGR 00637) by the Research and University Department of the Generalitat de Catalunya. *(Corresponding author: J. Abella)*

At the time of preparation of this paper all authors were with the Barcelona Supercomputing Center (BSC), Plaza Eusebi Guell, 1-3, 08034, Barcelona (Spain). At the time of submission F. Bas, G. Cabo and P. Benedicte are no longer with BSC. The email addresses for BSC members are sergi.alcaide@bsc.es, ilham.lasfar@bsc.es, feng.chang@bsc.es, francisco.fuentes@bsc.es, ramon.canal@bsc.es, juan.rodriguez3@bsc.es, and jaume.abella@bsc.es.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

I. Introduction

Systems-on-Chip (SoCs) used in a wide range of domains with safety requirements come equipped with Statistics Units (SUs), also known as performance monitoring units, devoted to monitor a variety of SoC events in the different Functional Unit Blocks (FUBs) integrated with the SoC. For instance, events monitored include those occurring in each computing core, such as processor cycles, instructions executed (and potentially their breakdown across types), in-core cache-related events, and the like; as well as those in shared resources such as bus/interconnect accesses per master, and stall cycles spent in different buffers along the communication path from the core to main memory to name a few.

SUs are increasingly used for the Verification and Validation (V&V) of the timing behavior of those safety-related systems when built upon SoCs, allowing some form of parallel execution (e.g., multicores, GPUs) [1]–[4]. In particular, SUs are one means to obtain evidence that interference experienced in the access to shared resources, such as shared caches, main memory, and peripherals, can be managed and kept within specific bounds [5]. Such interference observed during testing campaigns with stressful tests is used to validate that worst-case bounds estimated during the system design phase are not exceeded.

SUs available in most commercial SoCs provide indirect information about interference, such as number of accesses to shared resources, potentially broken down by core, as well as aggregated stall cycles in some queues and buffers, but without segregating stalls across those self-generated (e.g., due to previous requests of the task under analysis) and those produced by each other contender core (e.g., without identifying the offending task(s)).

To address this concern, enhanced SU support has been proposed and realized in some SoCs for the space domain to better master multicore interference by monitoring interference-specific information with the so-called SafeSU [6], [7]. The SafeSU measures stall cycles experienced by each core in the access to shared resources, and breaks those cycles down across self-generated ones and those generated by each one of the other cores. This information becomes extremely valuable to optimize task scheduling performed during system design, to test and debug timing behavior during validation of the system, and to implement informed safety measures to manage deadline overruns during operation since the offending task(s) can be easily identified, much in line with the safety net required by AMC 20-193 [8]. In the latter case, the system can be moved to a degraded operation mode dropping specific tasks (e.g., offending ones if non-critical), or an alternative pre-defined task schedule can be used where offending tasks do not run simultaneously with high-integrity tasks.

The SafeSU has been shown to be highly effective and accurate to monitor multicore interference in multicores relying on a single bus interconnect as the source of interference by using the master ID (core ID) of the requests to determine which core interferes with which other cores. However, with the increasing performance demands across domains, including the space domain, larger core counts are integrated, as well as multi-level cache hierarchies, hence with multiple hierarchical interconnects.

Protocols like AMBA Advanced High-performance Bus (AHB), Advanced eXtensible Interface (AXI), and the like, highly popular in space processors such as Gaisler's NOEL-V processor [9] (e.g., with an AHB bus between cores and the shared cache, and an AXI interface between such cache and DDR memory). These protocols use as identifier the interconnect-local identifier (e.g., master/slave ID for AHB, manager/subordinate for AXI). This implies that interfaces connecting shared caches with upper memory levels (e.g., DDR memory) use the cache ID, which does not allow to identify what core generated a specific request. Moreover, it is common releasing interconnects while requests are processed in other shared resources (e.g., DDR memory controller) to enhance parallelism. As a consequence, interconnects do not expose a significant fraction of the interference. Since the SafeSU uses interconnect-local IDs to ascribe interference to contenders, it is unable to monitor interference in multi-level interconnects, as well as interference occurring in other shared resources.

Lack of accurate interference observability could be addressed by devising SoC-wide solutions able to attach the core ID to each request in the SoC to track interference across interconnects and other shared resources, but this would require non-negligible changes across multiple Intellectual Properties (IPs). Alternatively, one could infer the core causing interference from available information in the SoC building ad-hoc solutions that monitor existing signals.

This work provides a solution in line with the latter – using information already available in the SoC – to measure and ascribe interference to cores. In particular, we propose a solution tailored to and integrated into a Gaisler's RISC-V space SoC [10], given that Gaisler's products are popular in the space missions from the European Space Agency, among other end users. We refer to this extended version of the SafeSU as SafeSU-2L (2L refers to the 2-level interconnect in the target SoC). While those solutions need to be devised on a platform-per-platform basis, they become minimally intrusive by just needed existing signals as input, without requiring any extra feature in existing IP. In particular, the contributions of this work are as follows:

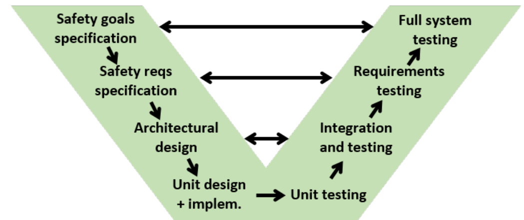


Fig. 1. Schematic of the usual V-model used for the development of safety-relevant electronic systems.

- We identify the different types (sources) of interference that can occur in a SoC with a 2-level interconnect where one of them has no visibility of the core IDs. In particular, we focus on a RISC-V space multicore SoC by Frontgrade Gaisler, the NOEL-XCKU-EX [10] (NOELSoC for short).
- Along with the identification of each of the types of interference, we provide details of how each one of them can be properly captured using the signals available in the SoC, which are routed to the extended SafeSU (the SafeSU-2L) that tracks multicore interference across all cores.
- We realize our SafeSU-2L in the 4-core NOELSoC implemented in a Xilinx Kintex UltraScale KCU105 FPGA, and evaluate it with stress benchmarks intended to generate corner performance cases, as well as with regular benchmarks showing typical scenarios. Our results prove that the SafeSU-2L is able to effectively estimate multicore interference during system validation phases and ascribe it to the corresponding aggressor core with high accuracy.

The rest of the paper is organized as follows. Section II provides some background on the NOELSoC and the SafeSU. Section III presents the SafeSU-2L. Section IV describes our evaluation framework. Section V provides a quantitative evaluation of the effectiveness of the SafeSU-2L. Related work is described in Section VI. Finally, Section VII summarizes this paper.

II. Background

This section introduces context motivating this work, the architecture of the NOELSoC and its characteristics particularly relevant for the SafeSU-2L, as well as the most relevant elements of the SafeSU for our work.

A. Safety-relevant System Development

System development for safety-relevant systems follows the classic V-model, as depicted in Figure 1. For this work, some specific stages are particularly relevant. The first one is the architectural design, where the system is decomposed into items taking over of different parts of the overall functionality and a subset of safety requirements. As part of this architecture, some safety measures are implemented to manage faults, such as, for instance, the ability to detect whether a task finishes its execution abnormally and reexecute it timely. In the context of our work, some safety measures may typically monitor whether execution time bounds are exceeded and, upon a deadline violation, an informed action needs to be taken. The SafeSU-2L provides input in the form of how much time each task has interfered each other task, particularly useful to select the most convenient safety measure (e.g., stop the offending task until affected tasks are successfully reexecuted).

As part of the architecture design and unit design and implementation, execution time bounds are estimated for each task to set a schedule that allows executing all tasks timely. While this schedule is intended to be correct by construction, worst-case execution time bounds must be validated during the testing campaigns (unit and integration testing) with relevant tests to collect evidence that all tasks will execute timely. Here, the SafeSU-2L plays a key role providing accurate information for diagnosis purposes in case of overrun by easing the process to identify the cause of the overrun. Even if there is not such an overrun, the information provided by the SafeSU-2L can be used to identify inefficiencies and optimize the task schedule (e.g., to integrate additional functionalities or enhance existing ones).

B. The NOEL-XCKU-EX Space SoC (NOELSoC)

The NOELSoC (see Figure 2) is a 4-core RISC-V multicore SoC targeting mainly the space domain. It includes 4 CAES Gaisler NOEL-V cores, each one implementing a local first level (L1) data cache and an L1 instruction cache, both 16KB 4-way set associative with LRU replacement. The L1 data cache is write-through so that all store operations are forwarded to the shared L2 cache, and no write allocate.

An AMBA Advanced High-performance Bus (AHB) bus, which we refer to as AHBcores, connects the cores to the shared L2 cache. The L2 cache space can be shared or partitioned across cores. The latter setup suits the needs of critical real-time tasks to provide time predictability. The L2 cache implements a write-back policy and write allocate to limit the number of (slow) DDR4 memory transactions. It is a 256KB 4-way cache with LRU replacement.

The L2 cache is connected to the DDR4 memory controller through an AMBA AHB bus, which we refer to as AHBmem. The memory controller can hold a single read request, and multiple write requests simultaneously. The AHBcores and AHBmem buses can each process a single request simultaneously.

The NOELSoC has been built in the context of a project with its components reaching at least a Technology Readiness Level (TRL) 6, even if in an FPGA, since radiation-hardened FPGAs are already commercial systems in the space domain. However, whether the SafeSU-2L will be deployed in ASIC implementations of the NOELSoC is unknown and will depend on whether end users demand its capabilities.

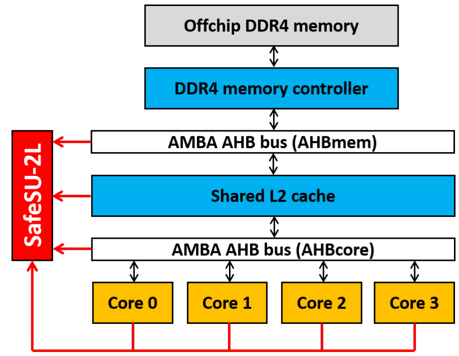


Fig. 2. Schematic of the NOELSoC extended with the SafeSU-2L. Cores are connected to a shared AHB bus and a shared L2 cache. An additional AHB bus is used to access DDR4 memory. The SafeSU-2L uses signals from both AHB buses, the L2 cache and the cores.

C. The SafeSU

The SafeSU [6], [7] implements several mechanisms to measure multicore interference broken down across contenders for each core: the Contention Cycle Stack (CCS) unit [11], a mechanism to set interference quotas that builds upon the CCS [12], and a mechanism to measure the maximum latency experienced per event in a bus [12]. Across those components, the SafeSU-2L enhances the CCS. Hence, we focus on the CCS and refer the interested reader to the SafeSU publications for further details of its other features.

The CCS of the SafeSU monitors the master IDs (core IDs) and the request/grant signals of an AMBA AHB bus to determine whether a core owns the bus (request signal enabled and grant signal enabled), or whether a core is delayed by another core X (request signal enabled and grant signal enabled for core X). However, if we consider, for instance, the AHBmem bus, we note that the only master ID corresponds to the L2 cache, and hence, the CCS is unable to detect any interference because L2 requests are not delayed by other masters, and moreover, the core owner of a given requests it is not known.

Note also that even memory space addresses themselves cannot be used as a form of core ID since there is not a direct correlation between addresses used and cores issuing the memory requests because no full memory partitioning across cores is feasible. This relates to multiple factors, being some of them the data exchange across applications potentially running in different cores (e.g., Task A runs in core 0 writing address range X, and later on Task B runs in core 1 reading address range X), as well as the access to services and shared buffers of the operating system. Overall, addresses cannot be used as a form of core ID for interfaces lacking those IDs.

Figure 2 illustrates the NOELSoC with the SafeSU-2L replacing (enhancing) the SafeSU. The SafeSU would only observe the AHBcores bus. Instead, as explained in the following section, the SafeSU-2L observes additional signals from other components to infer what core interferes with what other core.

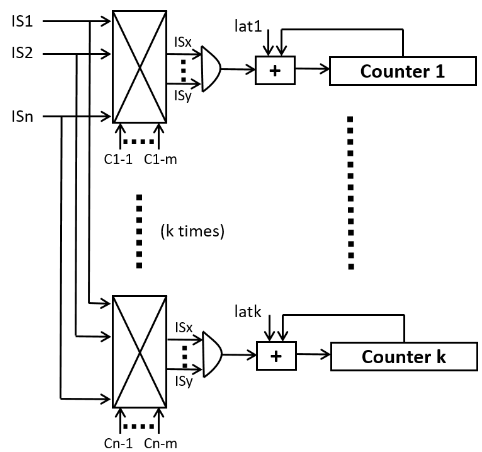


Fig. 3. Schematic of the general architecture of the SafeSU and SafeSU-2L, with n input signals (IS), k counters, and m signals controlling each counter. Control signals for each counter program the crossbar to select ISs. Whenever those ISs concur, the counter is increased with a pre-programmed value.

Figure 3 provides a schematic of the basic architecture of both, the SafeSU and the SafeSU-2L. In particular, the unit monitors n signals, indicated as IS (input signals) in the plot. Those signals differ across the SafeSU and SafeSU-2L, as explained later. The unit is programmed indicating what m signals determine when a counter needs to be increased. For instance, for counter 1 those signals are indicated as $C1-1$ up to $C1-m$. Those signals drive a crossbar routing the appropriate signals to an AND gate determining when the counter needs to be increased by a given amount provided by the lat parameter (e.g., $lat1$ for counter 19. The real design is a bit more complex to manage timing and signal transitions, but the schematic illustrates the main operation of the unit.

The main differences across the SafeSU and the SafeSU-2L relate to the specific signals monitored, as explaine later on, as well as to the number of counters needed in the SafeSU-2L to provide specific interference breakdowns.

III. SafeSU-2L

The CCS of the SafeSU is a module to measure interference in AHB interfaces, but it is not specifically tailored to the NOELSoC. In this section, we describe how the SafeSU has been tailored and extended to measure interference occurring in both AHB interfaces of the NOELSoC simultaneously. In particular, rather than considering only the AHB arbitration signals (i.e., which masters request the bus and which one is granted access), it also considers additional signals such as the following ones, routed from the respective components, namely the cores, the AHBcores bus, the L2 cache, and the AHBmem bus respectively:

- Core-related signals:
  - `CL1miss`: L1 data or instruction cache miss in the monitored core.
  - `CWBfull`: Whether the write buffer internal to the core is full and a new store cannot be stored in it.
  - `Cread`: Whether the core wants to send a read request to the L2 cache.
- Core-to-L2 bus (AHBcores for short) related signals:
  - `B1addr`: Address of the request in the AHB-cores bus.
  - `B1owner`: Owner of the AHBcores bus.
  - `B1req`: Cores requesting access to the AHB-cores bus.
- L2 cache-related signals:
  - `L2miss`: The L2 cache access misses in L2 cache.
  - `L2write`: The L2 cache access is a write operation.
- L2-to-MemCtrl bus (AHBmem for short) related signals:
  - `B2addr`: Address of the request in the L2-to-MemCtrl bus (AHBmem for short).
  - `B2read`: Whether the AHBmem bus is in use by a read request.

Note that, out of the signals above, the original SafeSU design only uses some of them related to the AHBcores, namely `B1owner` and `B1req`. Those signals allow measuring only a limited subset of the contention in the NOELSoC, as described later in this section.

In the remainder of this section, we introduce the different scenarios where interference occurs and how the signals above are used to measure it and ascribe it to the core causing it. The main components of the NOELSoC are shown in Figure 2. Note that, to ease explanations and illustration in the figure, we assume that we monitor Core 0 (C0 for short), but per-core signals and counters are set up for all cores.

Some of the types of interference identified next use a parameter expressed as a number of cycles (see Table I). Values observed matched a very large fraction of times the mode in all cases, with tiny variations of 1 or 2 cycles occurring sporadically due to spurious activity caused by unavoidable occasional interference from, for instance, peripheral activity, or infrequent casuistic caused by the interaction of multiple requests from the task under analysis. In this work, we focus on estimating the actual interference occurred rather than on estimating the worst interference possible, since our goal is validating the accuracy of the SafeSU-2L, and neither estimating or validating worst-case bound. Hence, we stick to central values for those latencies rather than to extreme (maximum or minimum) values, which would not allow assess whether discrepancies with actual interference are caused by the use of pessimistic/optimistic values or due to limitations of the proposed design.

TABLE I
Latencies for the NOELSoC.

Event	Latency	Description
$L2WmissLAT$	38	L2 write miss latency (end-to-end)
$L2RhitLAT$	8	L2 read hit latency
$L2RtravLAT$	6	L2 traversal lat. for L2 read miss
$L2RmissLAT$	24	L2 write miss latency (only DDR)

Moreover, while not observed in our experiments, in very unfrequent cases memory accesses could clash with DRAM refresh commands, which would lead to potentially higher access latencies. Using the maximum value observed in such a case could lead to overly pessimistic interference estimates, which would likely be of little use to assess the accuracy of the SafeSU-2L. In those cases, those measurements should be dropped to avoid contaminating the mean or, alternatively, resort to the statistical mode.

A. `INT1`: Read AHBcores Interference

The first type of interference (`Int1`) occurs when C0 wants to send a read request to the L2 cache, but the AHBcores bus is in use by another core. This occurs when `Cread` and `B1req` for C0 are set, and `B1owner` is not C0. This case matches that of the original CCS of the SafeSU for read operations.

$$Int1(Req) \mathrel{+}= \sum (Cread_i^{C0} \land B1req_i^{C0} \land \overline{\mathrm{B1owner}_i^{C0}}) \quad (1)$$

The equation above indicates that, for a given core request Req, there is interference every cycle i when the core wants to read from L2, and requests the bus, but there is a different bus owner.

B. `INT2`: Write AHBcores Interference

The second type of interference (`Int2`) occurs when store operations cannot access the L2 cache because it cannot admit further write requests, and hence, get stalled in the core, creating back-pressure in the core pipeline due to interference whenever they fill the write buffer of the core. If store accesses of a core hit in L2 and experience no interference, they never fill in the write buffer of the core. Instead, when the write buffer gets full (`CWBfull` is set) and generates contention, but no L2 cache miss is caused by the write requests of C0, then interference occurs. If L2 cache misses occur due to those write operations of C0, then the write buffer may get stalled during the time it takes to serve an L2 write cache miss ($L2WmissLAT$, 38 cycles in the NOELSoC as summarized in Table I). Other additional cycles beyond such latency to serve the miss can only correspond to interference.

$$Int2(Req) \mathrel{+}= \sum (CWBfull_i^{C0}) - L2WmissLAT \cdot (L2miss^{C0} \land L2write^{C0}) \quad (2)$$

The equation above indicates that, for a given core request Req, we count as interference every cycle that the write buffer is full, but we subtract $L2WmissLAT$ cycles if there is an L2 write miss of C0.

C. `INT3`: Read L2 Hit Cache Interference

L2 read accesses get stalled in the AHBcores bus by other read accesses issued by other cores. Hence, those cases are captured under the first type of interference (read AHBcores interference). However, write accesses from other cores can also cause interference on read requests. In particular, the third type of interference (`Int3`) corresponds to the case when L2 read hits reach the L2 cache (hence, no `Int1` interference), and take longer than they would take in a no-interference case. This occurs when there are write operations stored in internal L2 cache queues, and those write operations delay the service of the L2 read hit request.

$$Int3(Req) \mathrel{+}= (B1addr_i \neq B2addr_i) \cdot \Big(\sum (Cread_i^{C0} \land B1req_i^{C0} \land B1owner_i^{C0}) - L2RhitLAT\Big) \quad (3)$$

The equation above indicates that, for a given core request Req, there can only be interference if the address of the request in AHBcores bus and AHBmem bus differ, meaning that it does not miss in L2 cache. Then, there is interference during all cycles if the request is a read request that owns the AHBcores bus, but subtracting the $L2RhitLAT$ cycles required to serve the request if no interference is experienced at all (8 cycles for the NOELSoC).

D. `INT4`: Read L2 Miss Cache Interference

The fourth type of interference (`Int4`) is analogous to the previous one (`Int3`), but in the case that the read access to L2 cache is a miss. In other words, L2 read misses may experience interference in the L2 cache due to other requests pending. Such interference can be computed as the time elapsed since the read request arrives to the L2 cache through the AHBcores bus, until it appears again in the AHBmem bus, subtracting the minimum latency for that propagation to occur, $L2RtravLAT$ (6 cycles in the NOELSoC).

$$Int4(Req) \mathrel{+}= (B1addr_i = B2addr_i) \cdot \Big(\sum (Cread_i^{C0} \land B1req_i^{C0} \land B1owner_i^{C0}) - L2RtravLAT\Big) \quad (4)$$

The equation above indicates that, for a given core request Req, there can only be interference if the address of the request in AHBcores bus and AHBmem bus eventually match during the transaction (e.g., in cycle i), meaning that it misses in L2 cache. Then, there is interference during all these cycles if the request is a read request that owns the AHBcores bus, but subtracting the $L2RtravLAT$ cycles required to propagate the request to the AHBmem bus if no interference is experienced at all.

E. `INT5`: Read L2 Miss DDR4 Interference

The fifth type of interference (`Int5`) relates to interference caused by prior write transactions impacting later read transactions in the DDR4 memory. In the AHBmem bus, write transactions do not own the bus for more than one cycle since the information is stored in a memory controller buffer, similarly to the AHBcores bus and the internal L2 buffers. Instead, when the L2 performs a read transaction to memory, the transaction holds the bus until the data are finally retrieved. If there are no pending writes in the DDR4 memory buffer, the transaction always takes a similar number of cycles ($L2RmissLAT$, 24 for the NOELSoC). However, if there are writes pending in the DDR4 memory buffer, the memory bus can be held longer, similarly to the issue observed in the L2. As for `Int3` and `Int4`, interference is calculated by taking the cycles elapsed to serve the read operation subtracting the cycles it takes to serve it when there are no pending writes in the DDR4 memory buffer.

$$Int5(Req) \mathrel{+}= \sum(B2read_i) - L2RmissLAT \tag{5}$$

Since there can only be one in-flight read request, this is the one corresponding to C0. Hence, interference is measured as the time when the AHBmem bus is busy by a read request subtracting $L2RmissLAT$.

F. Comparison vs the SafeSU

As indicated before, the SafeSU only uses some AHBcores signals to measure interference. The SafeSU in general, and its CCS component in particular [11], was devised to measure interference in processors implementing an interconnect for the cores lacking support for split transactions, such as some early prototypes of the Gaisler's NGMP [13]. Hence, until a request is served, no further request can be issued in those designs, so `Int1` interference (extended to read and write requests) is the only source of interference.

In our target SoC, `Int1` interference only applies to read requests, hence the use of the $Cread_i^{C0}$ signal, but abundant read and write interference is not captured with this source only. Therefore, the other types of interference needed to be identified and tracked, which brings a significant contribution with respect to the original SafeSU design.

As illustrated in the Figure 3, in general, the SafeSU-2L is highly agnostic to the particular protocol implemented by the bus and, mainly, a wrapper is needed to drive the right signals to the SafeSU-2L. Hence, the core of the design may receive signals from different buses, potentially implementing different protocols, as well as signals from the cores or the cache memories, which eases portability and extendability of the unit. Some complexity is brought by the SafeSU-2L with respect to the SafeSU due to the extra signals needed, but on the other hand, some signals originally used by the SafeSU are no longer needed. Hence, despite signals routed to the SafeSU-2L change, the design of the module is only dependent on the number of signals routed, which in our specific instantiation in the NOELSoC did not change. Also, the minimum number of counters needed to monitor different interference sources is higher for the SafeSU-2L, but most of them could be mapped to already existing programmable counters available in the SafeSU. Given that more than 90% of the area of the SafeSU is devoted to the signal crossbars to drive input signals to each counter, and the programmable counters themselves, modifications needed in the logic to activate the counters (using AND, OR and XOR gates instead of only AND gates) can only cause a small area increase in circuitry accounting for less than 10% of the area itself.

In terms of scalability, as for the SafeSU, the SafeSU-2L may need an increased number of signals to monitor and counters to measure interference if the core count and interfaces to monitor increases. Yet, the relative area costs with respect to the overall SoC are not expected to increase noticeably.

G. Beyond the NOELSoC

The SafeSU-2L has been devised to fit the particular characteristics of the NOELSoC, which include specific cache write policies and degrees of request concurrency in the interconnects and the DDR memory controller. It is expected that, except for the latencies described in Table I, which need to be devised for the specific target device, most of the characteristics of the SafeSU-2L would remain the same for other space SoCs, as well as for a number of SoCs across other domains.

In particular, cache write policies, with a L1 write-through data cache and an L2 write-back cache, are expected to be the best tradeoff for space SoCs since they allow using only parity protection in L1 caches (i.e., data can be invalidated upon an error and retrieved from L2) to manage frequent errors in a harsh environment, and avoid frequent DDR memory accesses (i.e., L2 does not propagate every single write operation to memory by being write-back). Hence, interference propagation across the different shared components in the SoC should remain similar, hence with similar types of interference to the ones described before, and measurable with analogous means. If, instead, highly different cache policies were used (e.g., a write-back L1 data cache), then the types of interference would need review. Note, however, that interference for write-back caches is shown to be measurable with the SafeSU-2L since this is the write policy for the L2 cache in the NOELSoC, so at least a significant part of the rationale above should still hold.

Larger SoCs with more core counts and/or additional cache levels and interconnects are expected to be manageable for the SafeSU-2L, requiring limited changes in the SafeSU-2L as long as they are managed with similar interfaces to the NOELSoC only varying the number of hierarchical interconnect levels in the communication path from the cores to memory. However, if highly different architectures are devised, such as mesh-based

SoCs and/or memory controllers enabling a high degree of request concurrency, the SafeSU-2L may need to be deeply revised to consider the new SoC characteristics. Still, signals comparable to the ones used in our work, such as those related to interconnect interfaces and some related to shared cache behavior, are expected to expose all interference and allow ascribing it to the actual sources of interference.

Large memory transactions and other comparable activity caused by accelerators and I/O devices is visible at software level (i.e. it is explicit in the source code), and it is typically driven through the operating system and/or the hypervisor in safety-relevant systems, with system software providing partitioning and scheduling mechanisms to make those effects have a known effect and already accounted for in the tasks response time. Therefore, neglecting those effects in the estimates provided by the SafeSU-2L is a safe assumption as long as system software manages those activities explicitly. Therefore, multicore interference estimates provided by the SafeSU-2L are expected not to be affected by those coarse-grain activities in safety-relevant systems, and would only be challenged in systems where such activity is not explicitly controlled by the system software.

IV. Evaluation Framework

We have three groups of experiments, using two different sets of benchmarks, the μbenchmarks and the TACLe Benchmarks [14] or taclebench for short. μbenchmarks are tiny programs we have developed that perform an intense and sustained type of activity, such as sending sustained requests of one type (read or write) that access a specific level of the memory hierarchy, which is achieved by accessing properly sized vectors that are traversed with a stride matching the cache line size used in all cache memories. For instance, one such μbenchmarks sends sustained write requests that hit in the L2 cache. Regarding taclebench, they are a set of benchmarks of different sizes and complexity, particularly devised to evaluate critical real-time embedded systems, offering a wider variety of multicore interference scenarios, and not necessarily high interference ones like those of the μbenchmarks. Since taclebench are self-contained, i.e., input data is embedded into the benchmarks themselves, they are easy to deploy on a bare-metal setup.

We consider three types of workload, in all of them running the benchmarks on bare-metal. In particular, those types of workloads are as follows:

- The core under analysis (C0) executes a μbenchmark 1,000 times so that its execution time is, at least above 50,000 cycles (in many cases between 1 and 3 million cycles). The rest of the cores (C1-C3) also execute a μbenchmark each (either the same across contender cores or different) in an infinite loop so that they generate interference during the complete execution of the benchmark in the core under analysis.
- The core under analysis (C0) executes a taclebench 1,000 times. The rest of the cores run μbenchmarks in an infinite loop, as for the previous workload type.
- The core under analysis (C0) executes a taclebench 1,000 times. The rest of the cores run the very same taclebench as C0, but in an infinite loop.

TABLE II
Mixed μbenchmark contender groups.

NAME	C1	C2	C3
READL2HIT	READL2HIT	READL2HIT	READL2HIT
READL2MISS	READL2MISS	READL2MISS	READL2MISS
MIX1	READL2HIT	WRITEL2HIT	READL2MISS
MIX2	WRITEL2HIT	READL2MISS	WRITEL2MISS
MIX3	READL2MISS	WRITEL2MISS	READL2HIT
MIX4	WRITEL2MISS	READL2HIT	WRITEL2HIT
WRITEL2HIT	WRITEL2HIT	WRITEL2HIT	WRITEL2HIT
WRITEL2MISS	WRITEL2MISS	WRITEL2MISS	WRITEL2MISS

The decision of running the benchmark in the core under analysis 1,000 times has been taken solely for the purpose of making the usual types of interference in each case become the dominant ones, hence easing the reasoning on the results. We could have run each benchmark just once. However, unfrequent request time alignment could have occurred in some runs so that, while the SafeSU-2L would deliver precise results, those would not be obvious to analyze.

The μbenchmarks used are as follows:

- READL1HIT: Performs 64 consecutive loads that hit in the L1 data cache.
- READL2HIT: Performs 64 consecutive loads that miss in the L1 data cache and hit in the L2 cache.
- READL2MISS: Performs 64 consecutive loads that miss in both the L1 data cache and the L2 cache.
- WRITEL1HIT: Performs 64 consecutive stores that hit in the L1 data cache.
- WRITEL2HIT: Performs 64 consecutive stores that miss in the L1 data cache and hit in the L2 cache.
- WRITEL2MISS: Performs 64 consecutive stores that miss in both the L1 data cache and the L2 cache.

Those μbenchmarks have been designed so that they cannot generate additional cache misses on each other when running simultaneously by properly setting access patterns and footprints. Analogously, we have verified that taclebenchs do not suffer additional L2 cache misses when running together with some μbenchmarks, and they neither experience any relevant extra number of L2 cache misses when running against multiple copies of themselves on the NOELSoC.

Those 6 μbenchmarks are used as task under analysis for the first workload type. For the first and second workload types, we exclude the L1HIT μbenchmarks as contenders since they do not create any interference and use the remaining 4 μbenchmarks (L2HIT and L2MISS ones). With those 4 μbenchmarks we generate 8 contender combinations. Out of those 8 combinations, 4 of them correspond to using identical μbenchmarks across contender cores (e.g., cores C1 to C3 all run WRITEL2HIT in an infinite loop). The remaining 4 combinations are arbitrary combinations of different μbenchmarks in each core, as shown in Table II. In the different plots, workloads are sorted taking into account the amount of write DDR4 contention of the workloads, as this is particularly useful in one of the plots shown later. Hence, homogeneous workloads with read contenders are shown first, then MIX1, then the remaining of the mixes, and finally workloads with write contenders.

Note that benchmarks in a workload are mapped to independent memory regions, so not reusing any data or code across them. Also, experiments are run starting benchmarks synchronizedly (only very few cycles of difference between their start times), but their execution is not explicitly controlled in any manner, meaning that small variations already discussed can lead to different order to process the memory requests from different cores in the L2 cache and DDR4 memory.

V. Evaluation Results

A. μbenchmarks vs μbenchmarks

1. Accuracy

To assess the accuracy of the SafeSU-2L to estimate the overall interference, we have run 48 workloads considering the 6 μbenchmarks as task under analysis, and for each one, the 8 workloads described before (4 with identical μbenchmarks across contender cores, and 4 with the mixes described before). Figure 4 shows the addition of execution times in isolation and estimated interference with the SafeSU-2L w.r.t. execution times observed with the corresponding workloads, hence reflecting the accuracy of the SafeSU-2L estimating interference. Note that, in the ideal case, if interference estimated matched real interference, we would get all values at 100% in the plot. Hence, inaccuracy is obtained as the deviation with respect to 100%.

This particular comparison has been selected because, for instance, READL1HIT experiences almost no interference[1], and hence, few cycles of variation between estimated and real interference could produce very large relative variations if we compare only interference. Instead, aggregating interference with execution time in isolation allows considering the relative impact of estimated interference in the overall execution time.

For instance, when running READL1HIT in C0 and READL2HIT as contender in C1-C3, execution time is above 65,000 cycles, real interference 204 cycles, and estimated interference 225 cycles. Hence, a deviation of 21 cycles w.r.t. an execution time above 65,000 cycles is around 0.03%. Instead, if we only compare interference, 21 cycles w.r.t. 204 is in the order of 10%, so 2 or 3 orders of magnitude higher. Conversely, in cases with large interference, such as READL2MISS in C0 versus READL2MISS as contenders, the execution time in isolation is 2,219,229 cycles, the measured interference 5,570,320 cycles, and the interference estimated with the SafeSU-2L is 5,626,989. Hence, the order of magnitude of any deviation comparing total multicore execution time or only multicore interference remains the same (e.g., 0.7% vs 1.0% in this example).

Overall, results in figure 4 show that the SafeSU-2L is highly precise in estimating interference with an average and geometric mean deviation of 1.1% and 0.4% respectively, being 4.8% and 5.7% the largest underestimate and overestimate respectively, with 35 out of 48 workloads having a deviation below 1%.

Those minor deviations naturally relate to those characteristics of the actual hardware platform not captured by the formulation used for the different interference types. For instance, some internal queues in the L2 cache and the DDR controller may introduce some small time variations depending on their state when new requests arrive. Also, DDR memory refreshes may alter time measurements depending on when they occur. Since those effects are not modelled due to the complexity to model those details – if at all possible, some inaccuracies can occur. Yet, as shown, those are small since major effects are properly captured by the SafeSU-2L.

To further validate that execution time increases are caused only due to multicore interference, we repeated small subsets of the experiments in a simulated environment were no variations at all could be experienced. We cross-validated those results against those running the experiments in the FPGA and found no relevant variations in terms of miss counts, L2 accesses and bus transaction counts. Differences were only observed in cycle counts due to the different memory latencies in both setups. In any case, all trends and conclusions held anyway, and allowed us conclude that the only relevant source of execution time variation with respect to the setup in isolation is multicore interference.

2. Interference types

To provide further insight into the results, we also provide the absolute interference estimated broken down across sources of interference in accordance with the 5 categories described before (from `Int1` to `Int5`). Results are shown in Figure 5. As expected, the READL1HIT μbenchmark does not experience any meaningful interference since it hits its local L1 data cache sustainedly.

WRITE μbenchmarks only experience relevant `Int2` interference, and any other type of interference has negligible impact. This relates to the fact that WRITE benchmarks, upon experiencing some delay in their way to L2 cache or DDR4 memory, quickly saturate their

[1]This is the expected behavior for this benchmark since it barely accesses shared resources, and hence, can only be negligibly be affected by interference from others.

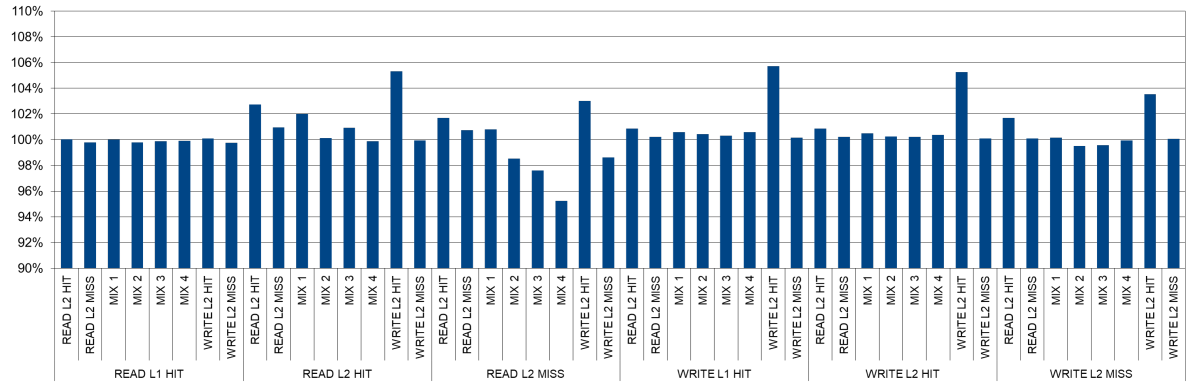


Fig. 4. Interference estimation accuracy: Interference estimated with the SafeSU-2L vs real interference (μbenchmarks vs μbenchmarks). The x-axis shows the task under analysis (bottom) and the set of contenders (see Table II for details). 100% means fully accurate estimates, higher values overestimation, and lower values underestimation. Note that the scale ranges between 90% and 110%.

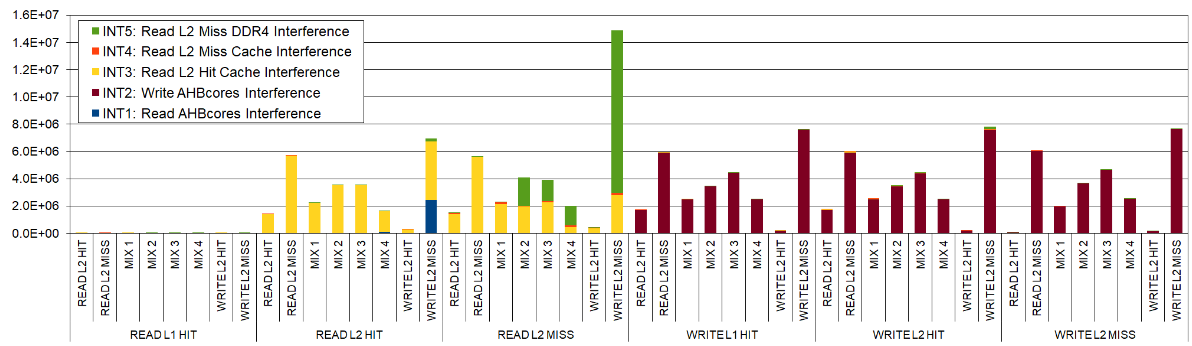


Fig. 5. Absolute interference (in cycles) broken down across those types presented in Section III, for the same workloads described in Figure 4 (μbenchmarks vs μbenchmarks).

write buffer, hence generating back-pressure in the core pipeline.

READL2HIT and READL2MISS μbenchmarks experience significant `Int3` interference across workloads, which corresponds to the interference experience within the L2 cache due to other requests being served in the L2 cache. `Int5` interference is only relevant for READL2MISS when there are write requests being served in the DDR4 memory. This is the case for WRITEL2MISS as well as for workloads MIX2, MIX3, and MIX4, and hence why those workloads produce relevant interference of this type.

`Int1` interference is only relevant in one case where the task under analysis performs sustained read hits in L2 (READL2HIT) that would be served by the L2 directly, and contenders generate a very high number of L2 requests that take longer to be served, which is the case of the WRITEL2MISS benchmark. In this case, those write memory requests produce this type of interference on READL2HIT. Note that `Int1` interference is the only type of interference that would be captured by the original SafeSU if integrated *as it is* in the NOELSoC.

Finally, while `Int4` interference is observed in many workloads, it is generally very low, and hence, it could be possible to drop the logic to measure such interference with a very low impact on accuracy.

3. *Interference core source*

Figure 6 shows the breakdown of interference experienced by C0 across contender cores normalized w.r.t. the total interference experienced.

First, we note that workloads with homogeneous contenders (all except mixes) generally show that the interference caused by each of the contender cores (C1-C3) is estimated as highly homogeneous, as expected. Still, there are a few exceptions. For instance, for READL1HIT in C0 and WRITEL2HIT as contender, the distribution is not homogeneous. In this case, as explained before, the amount of interference is tiny, and hence, small variations of a few tens of cycles create large relative discrepancies.

Second, in several cases, such as READL2HIT in C0 and WRITEL2HIT as contender, show lower interference coming from C1 than from C2 and C3. Similarly, in the case of WRITEL2MISS in C0 and READL2HIT as contender, interference caused by contenders is highly biased. We note that in these cases the absolute interference is very low (see Figure 5), and hence, small deviations in absolute terms cause large relative deviations. Moreover, these cases relate to the order of arrival of requests. The different overlapping of C0 requests with C1-C3 requests in some spurious cases that may repeat periodically can cause small absolute bias but high in relative terms.

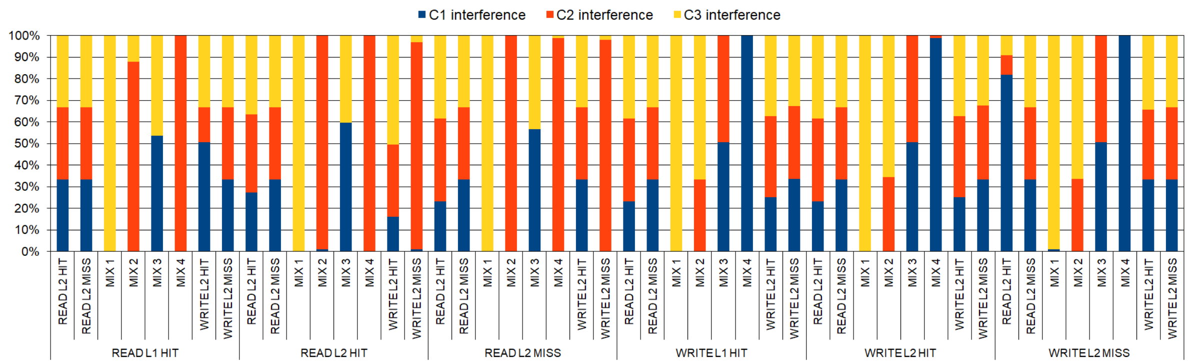


Fig. 6. Interference breakdown across contender core source (μbenchmarks vs μbenchmarks) for the same workloads described in Figure 4.

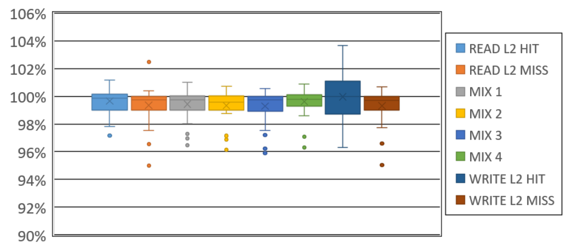


Fig. 7. Interference estimated with the SafeSU-2L vs real interference (taclebenchs vs μbenchmarks). Each column corresponds to a boxplot for all taclebench when run against the workload indicated in the legend. 100% means fully accurate estimates, higher values overestimation, and lower values underestimation. Note that the scale ranges between 90% and 106%.

B. Taclebenchs vs μbenchmarks

After assessing the SafeSU-2L in the context of stressful benchmarks only, in this section, we show the accuracy of the SafeSU-2L when using representative benchmarks (taclebenchs) subject to stressful interference conditions (against μbenchmarks).

We have summarized the data of the 29 benchmarks with boxplots, shown in Figure 7, for each of the contender workloads. We observe that, as for the case of μbenchmarks vs μbenchmarks, interference estimates have an accuracy typically within 1% the real interference, and deviations are scarce and, at most, up to 5%. The only (minor) difference we observe across contender workloads is that the bulk of the distribution has a slightly wider spread for the WRITEL2HIT case. However, as shown before, those cases where contenders run WRITEL2HIT typically are low-interference cases where variations are less relevant since their absolute impact is limited. In fact, small absolute variations in those cases have a larger relative impact than in other cases, and hence the wider spread in terms of accuracy for the WRITEL2HIT case.

C. Taclebenchs vs taclebenchs

For completeness, Figure 8 shows the results of running taclebenchs against 3 copies of themselves in the other cores. As expected, since interference created by public benchmarks with a small footprint, is lower than that created by stressful benchmarks (typically between 1% and 21%), the relative impact of inaccuracy decreases. Hence, deviations of the interference predictions restrict to up to 1.6% in the worst case, and 0.5% on average.

Overall, our experiments show that the SafeSU-2L provides accurate estimates for *actual* interference in the NOELSoC for a wide variety of multicore interference scenarios, which is of high relevance for the V&V of critical real-time systems, such as those in the space, avionics, and automotive domains, on top of this commercial SoC. Whether it could be used for worst-case execution time estimation is left for future work.

D. Hardware Costs

As reported in the original SafeSU work [6], it requires around 5,000 LUTs, which compares against the 192,400 LUTs needed by the cores only. In particular, we have collected detailed measurements, and the number of LUTs required for the original SafeSU are 4,673 LUTs and 2,545 registers, which represent 2.4% and 2.6% of the area of the 4 cores respectively. If we consider the full SoC, such relative cost roughly halves due to the space needed by the L2 cache and peripheral logic.

The SafeSU-2L requires additional 1,349 LUTs and 1,094 registers, hence increasing the size of the SafeSU by 29% and 43% respectively, but increasing the size of the design by only 0.7% and 1.2% respectively when compared against the 4 cores in the SoC, and by half of that (0.4% and 0.6%) when considering the whole SoC.

Overall, the hardware cost required by the SafeSU-2L to account for 4 additional types of interference w.r.t. the SafeSU is very low since it reuses most of the logic and registers already existing in the SafeSU. Most of the induced costs relate to the registers needed to store interference values broken down across interference types instead of a single aggregated value.

VI. Related Work

Different timing validation and monitoring approaches for critical real-time applications on COTS multicores building on SUs have been devised in recent years. Those

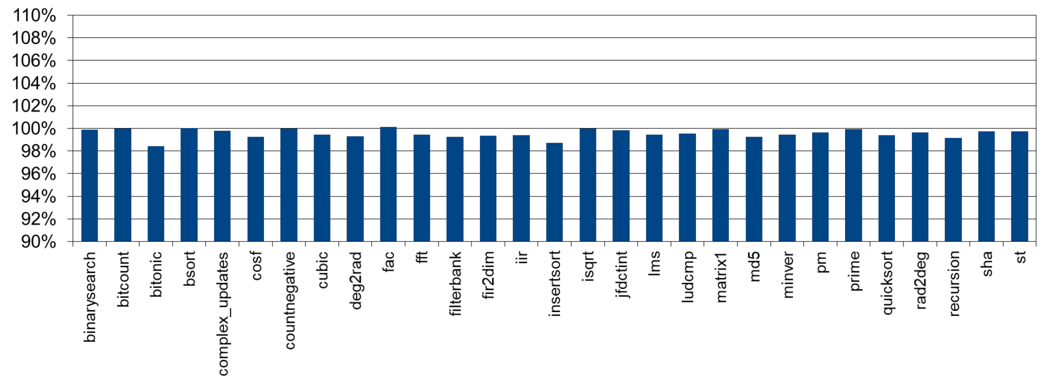


Fig. 8. Interference estimation accuracy: Interference estimated with the SafeSU-2L vs real interference (taclebenchs vs taclebenchs). The x-axis shows the benchmark used as both, task under analysis and for the 3 contenders. 100% means fully accurate estimates, higher values overestimation, and lower values underestimation. Note that the scale ranges between 90% and 110%.

approaches have reached different degrees of maturity. For instance, authors in [5] already offer a commercial service for multicore timing analysis on COTS targeting avionics, space, and automotive systems. Such an approach builds upon the development of tests to verify timing requirements, where each such test builds upon specialized micro-benchmarks whose correctness is assessed against event counts obtained through the SU.

Some techniques aim at estimating multicore interference bounds based on access counts to shared hardware resources and tests that provide estimates of the highest interference latency that can be experienced by access in each such shared resource. Those techniques target different platforms such as the Infineon AURIX TC27x family for automotive [1], [15], the Arm Juno board based Arm big.LITTLE multicores similar to those in the Xilinx Zynq UltraScale+ family considered for the avionics domain and in the Qualcomm SnapDragon family considered for the telecommunications domain [2], Cobham Gaisler NGMP LEON4 SoC for the space domain [3], and NXP P4080 SoC for the avionics domain [16]. Probabilistic analysis has also been explored on top of SUs to estimate multicore interference recently showing potential for the probabilistic avenue [17], [18].

SUs have also been used to analyze the timing behavior of highly-complex SoCs such as the NXP T4240, focusing on the L2 cache sharing [19] as well as on the coherence protocol [20]. Quantitative evidence shows that those high-performance complex SoCs bring plenty of performance along with some risks for safety-critical tasks due to their complexity.

Other techniques build upon SUs to monitor interference at runtime and take corrective actions to bound it. These techniques have been integrated on top of NXP P4080 and T2080 SoCs for avionics, the Arm Juno board relevant for avionics and telecommunications, and some virtual platforms such as Arm FVP or QEMU [4], [21]. Memguard and other similar techniques [22], [23] aim at controlling the bandwidth used by each task at runtime so that specific bounds are not violated. These software-only techniques build upon SUs to collect information relevant to estimate bandwidth usage.

Overall, SUs have already become the cornerstone on which abundant techniques, from research artifacts to commercial services, build to deliver multicore timing guarantees and thus, need appropriate SUs to easily build certification arguments on top to meet the most stringent safety integrity/assurance levels in space [24], avionics [25] and automotive [26].

VII. Conclusions and Future Work

Precise monitoring of multicore interference is key for the design, V&V, and optimization of safety-relevant SoCs, as well as to implement safety measures that must diagnose the cause of deadline overruns to take corrective actions. The SafeSU is a statistics unit providing appropriate features for those purposes but is only suitable for SoCs where interference occurs in a single interconnect where requests include the core ID of the core generating them. However, increasingly complex SoCs with multi-level interconnects do not meet those requirements since interference occurs in multiple interconnects instead of one, and at least one of them lacks core IDs that allow easily tracking aggressors and victims of interference.

This paper presents the SafeSU-2L that proves to be effective in monitoring interference and ascribe it to the appropriate aggressors in multicores with multi-level interconnects, including one without core IDs. In particular, we realize the SafeSU-2L in a commercial RISC-V multicore SoC for the space domain and show its effectiveness ascribing interference caused and experienced to cores. The SafeSU-2L, whose hardware cost is negligibly higher than the baseline SafeSU, becomes a key component for safety-relevant multicore SoCs.

Part of our future work consists of devising a protocol to measure the latencies of some events so that they can be programmed, easing the use of the SafeSU-2L in multiple incarnations of the NOELSoC (e.g., with different DDR memories with varying latencies, as part of FPGAs and ASICs, etc.) and in other SoCs. The approach behind the SafeSU-2L and its design span far beyond the space SoC used in this work. Hence, we believe that it can be adopted in different domains, with particular attention for the automotive domain due to the existing efforts to develop high-performance RISC-V automotive SoCs.

Acknowledgement

This work has been funded by the European Commission's Horizon Chips Joint Undertaking (JU) framework, under grant agreement No 101140087 (project SMARTY) and by PCI2024-153411 co-financed by MICIU/AEI /10.13039/501100011033.

REFERENCES

[1] E. Mezzetti, L. Barbina, J. Abella, S. Botta, and F. J. Cazorla, "Aurix tc277 multicore contention model integration for automotive applications," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 1202–1203.

[2] G. Fernandez, F. J. Cazorla, J. Abella, and S. Girbal, "Assessing time predictability features of arm big. little multicores," in *30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2018, pp. 258–261.

[3] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. Cazorla, "Bounding resource-contention interference in the next-generation multipurpose processor (NGMP)," in *8th European Congress on Embedded Real Time Software and Systems (ERTS^2)*, 2016.

[4] J. Nowotsch, M. Paulitsch, D. Bühler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity enforcement," in *26th Euromicro Conference on Real-Time Systems*, 2014, pp. 109–118.

[5] F. Cazorla, I. Broster, J. Abella, C. Evripidou, E. Mezzetti, and G. Bernat, "Understanding interference in critical multicore systems," in *24th Data Systems In Aerospace Conference (DASIA)*, 2019.

[6] G. Cabo, F. Bas, R. Lorenzo, D. Trilla, S. Alcaide, M. Moretó, C. Hernández, and J. Abella, "SafeSU: an extended statistics unit for multicore timing interference," in *2021 IEEE European Test Symposium (ETS)*, 2021, pp. 1–4.

[7] G. Cabo, S. Alcaide, C. Hernández, P. Benedicte, F. Bas, F. Mazzocchetti, and J. Abella, "SafeSU-2: a safe statistics unit for space MPSoCs," in *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2022, pp. 1085–1086.

[8] European Union Aviation Safety Agency (EASA), *AMC 20-193 Use of multi-core processors (MCPs)*, 2022.

[9] Frontgrade Gaisler, "NOEL-V Processor," 2023. [Online]. Available: https://www.gaisler.com/index.php/products/processors/noel-v

[10] N.-J. Wessman, F. Malatesta, J. Andersson, P. Gomez, M. Masmano, V. Nicolau, J. L. Rhun, G. Cabo, F. Bas, R. Lorenzo, O. Sala, D. Trilla, and J. Abella, "De-RISC: the first RISC-V space-grade platform for safety-critical systems," in *2021 IEEE Space Computing Conference (SCC)*, 2021, pp. 17–26.

[11] J. Jalle, M. Fernandez, J. Abella, J. Andersson, M. Patte, L. Fossati, M. Zulianello, and F. J. Cazorla, "Contention-aware performance monitoring counter support for real-time mpsocs," in *2016 11th IEEE Symposium on Industrial Embedded Systems (SIES)*, May 2016, pp. 1–10.

[12] J. Cardona, C. Hernandez, J. Abella, and F. J. Cazorla, "Maximum-contention control unit (mccu): Resource access count and contention time enforcement," in *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019, pp. 710–715.

[13] Cobham Gaisler, "GR740 Quad-Core LEON4 SPARC V8 Processor," 2019. [Online]. Available: https://www.gaisler.com/index.php/products/components/gr740

[14] H. Falk et al., "TACLeBench: A benchmark collection to support worst-case execution time research," in *WCET Workshop*, 2016.

[15] E. Díaz, E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla, "Modelling multicore contention on the aurixtm tc27x," in *Proceedings of the 55th Annual Design Automation Conference*, ser. DAC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3195970.3196077

[16] J. Nowotsch and M. Paulitsch, "Leveraging multi-core computing architectures in avionics," in *European Dependable Computing Conference (EDCC)*, 2012.

[17] A. Mascareñas González, Y. Bouchebaba, and L. Santinelli, "Multicore shared memory interference analysis through hardware performance counters," in *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, Toulouse, France, Jan. 2020. [Online]. Available: https://hal.science/hal-02446031

[18] S. Vilardell, I. Serra, R. Santalla, E. Mezzetti, J. Abella, and F. J. Cazorla, "Hrm: Merging hardware event monitors for improved timing analysis of complex mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3662–3673, 2020.

[19] J. Fish and A. Bognar, "Investigation of l2-cache interferences in a nxp qoriq t4240 multi-core processor," in *Architecture of Computing Systems – ARCS 2019*, M. Schoeberl, C. Hochberger, S. Uhrig, J. Brehm, and T. Pionteck, Eds. Cham: Springer International Publishing, 2019, pp. 183–194.

[20] N. Sensfelder, J. Brunel, and C. Pagetti, "On How to Identify Cache Coherence: Case of the NXP QorIQ T4240," in *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Völp, Ed., vol. 165. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 13:1–13:22. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/12376

[21] S. Girbal, J. Le Rhun, and H. Saoud, "METrICS: a measurement environment for multi-core time critical systems," in *9th European Congress on Embedded Real Time Software and Systems (ERTS^2)*, 2018.

[22] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. 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)*, 2013, pp. 55–64.

[23] R. Mancuso, R. Pellizzoni, N. Tokcan, and M. Caccamo, "WCET Derivation under Single Core Equivalence with Explicit Memory Budget Assignment," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), M. Bertogna, Ed., vol. 76. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 3:1–3:23. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7168

[24] European Cooperation for Space Standardization, "ECSS-Q-ST-80C, space product assurance: Software product assurance," 2009.

[25] RTCA and EUROCAE, *DO-178C / ED-12C, Software Considerations in Airborne Systems and Equipment Certification*, 2011.

[26] International Standards Organization, *ISO/DIS 26262. Road Vehicles – Functional Safety*, 2009.

Francisco Bas , MSc (2022), is a Hardware Engineer at Frontgrade Gaisler Technologies, specializing in RISC-V core development for space applications. He works on designing and optimizing RISC-V processors to ensure high performance and reliability in demanding environments, such as space missions. Francisco's work contributes to advancing the use of RISC-V in critical systems, addressing challenges related to performance and reliability in space-grade hardware.

Sergi Alcaide (PhD, 2023) is a post-doc of the High-Performance Embedded Systems (HPES) Lab at BSC where he manages a team of engineers and co-Director of PhD Students and leads the technical effort of BSC in several European Projects such as NimbleAI and SMARTY. Sergi's research interests includes RISC-V architectures for safety-critical and security systems.

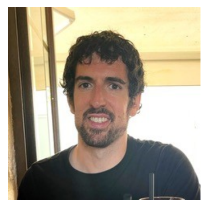
Guillem Cabo , MSc (2020), is a Hardware Engineer at Frontgrade Gasiler Technologies. He works on the design and development of RISC-V and SPARC System-on-Chips (SoCs) for space and other high-reliability applications. Guillem's contributions help ensure these systems perform reliably in demanding environments, addressing challenges such as resource contention in multicore and heterogeneous architectures.

Ilham Lasfar is a PhD student at the High-Performance Embedded Systems (HPES) Lab at the Barcelona Supercomputing Center (BSC), Spain, in collaboration with the Universitat Politecnica de Catalunya (UPC). Prior to starting her PhD, she worked for three years as a Research Engineer, specializing in hardware design and RTL development for real-time embedded systems. Her current research focuses on post-quantum cryptography (PQC) accelerators for safety-critical real-time applications.

Feng Chang (PhD, 2011) is the recognised researcher of the High-Performance Embedded Systems (HPES) Lab at BSC with around 60 researchers and engineers. Feng joins a number of projects on the use of AI, RISC-V architectures, and advanced software frameworks in safety-, security- and mission-critical systems.

Francisco Fuentes (Master in Telecommunications, 2022) is a computer science researcher of the High-Performance Embedded Systems (HPES) Lab at BSC (Barcelona, Spain). Francisco interests include research and development of hardware traffic injectors in safety-critical environments within the RISC-V ecosystem, fundamental work in his PhD thesis in progress.

Ramon Canal (PhD 2004) is a Professor UPC and part-time researcher at BSC. Previously, he worked at Sun Microsystems, was a Fulbright visiting scholar at Harvard University and a visiting professor at the University of Cyprus. His interests expand from reliability (process variation, soft-errors) to security at the architecture and circuit levels.



Pedro Benedicte (PhD, 2022) is a computer architect at Graphcore, where he has worked for over five years specifying and optimizing next-generation AI hardware. He previously spent six years at the Barcelona Supercomputing Center (BSC) conducting research on high-performance, critical real-time systems, and also completed an internship at Arm, contributing to advanced processor design initiatives.

Juan Carlos Rodriguez is a Research Engineer in the High-Performance Embedded Systems (HPES) Lab at the Barcelona Supercomputing Center (BSC) in Barcelona, Spain. He has over 7 years of experience designing software for embedded systems in both bare-metal and Linux environments. In the past 3 years, his work has focused on developing RTL IPs and their integration with RISC-V architectures.

Dr. Jaume Abella (PhD, 2005) is the co-Director of the High-Performance Embedded Systems (HPES) Lab at BSC with around 60 researchers and engineers, and invited professor at UPC, Barcelona (Spain). Jaume drives a number of projects on the use of AI, RISC-V architectures, and advanced software frameworks in safety-, security- and mission-critical systems. Jaume has created a spinoff company, and authored around 300 publications in top conferences and journals, and 15 patents.