SafeSU: an Extended Statistics Unit for Multicore Timing Interference

Guillem Cabo[†,‡], Francisco Bas[†,‡], Ruben Lorenzo[†], David Trilla[†],
Sergi Alcaide[†], Miquel Moretó[†], Carles Hernández[†,*], Jaume Abella[†]
[†]Barcelona Supercomputing Center (BSC)
[‡]Universitat Politècnica de Catalunya (UPC)
∗Universitat Politècnica de València (UPV)

***Abstract*—Statistics units (SUs) in MPSoCs are becoming increasingly used for the (1) verification and (2) validation of multicore timing interference, as well as for (3) deploying safety measures in safety-related real-time systems. However, existing SU extensions to manage multicore timing interference have neither been integrated together nor deployed in commercial MPSoCs.**

This paper presents the realization of the Safe Statistics Unit (SafeSU for short), which smartly integrates existing solutions for multicore timing interference verification, validation and monitoring, and is in turn integrated in commercial space-graded RISC-V and SparcV8 MPSoCs. Our evaluation illustrates the operation of the SafeSU, and paves the way for a thorough validation prior to reaching commercialization and being offered as open source IP.

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

I. Introduction

MultiProcessor Systems-on-Chip (MPSoCs) are becoming the de-facto standard for safety-critical real-time systems in multiple domains due to their capability to provide increased performance within limited power and area envelopes. Cobham Gaisler's LEON family for the space domain; NXP T/P/LS families and Xilinx Zynq family for the avionics and railway domains; and Infineon AURIX family for the automotive domain are examples of those MPSoCs [12].

However, safety-critical systems must undergo strict verification and validation (V&V) processes, as well as include appropriate safety measures to manage faults during operation. This is achieved by deploying a number of hardware features related to MPSoC controllability and observability. While such support has been successfully delivered in single-core SoCs, MPSoCs bring a new set of performance challenges due to their hardware shared resources, which lead to timing interference across cores. Such timing interference needs to be properly mastered to meet performance requirements in domain-specific safety standards.

Existing MPSoCs, even if explicitly devised for safety-critical applications (e.g. the Infineon AURIX processor family), provide limited support to manage multicore timing interference. In particular, those MPSoCs often provide Statistics Units (SUs) with event counters related to access counts and types in the different functional unit blocks (FUBs), and aggregated stall cycles in some particular FUBs. Some of those MPSoCs also provide powerful debug ports that, coupled with appropriate debug interfaces and software frameworks (e.g. Aurora, CodeWarrior, Lauterbach), allow tracing detailed events from the platform. However, those SUs and trace-related information have a number of limitations:

- Timing interference information provided hardly allows ascribing what core interferes what other core and by how many cycles (e.g. stalls may be simply provided aggregated).
- Means to control such interference are limited – if any (e.g. interrupts upon reaching pre-programmed event counts).
- Trace support requires interfaces and/or host computers only available during V&V, not during operation, thus not allowing to build safety measures on top.

A number of solutions have been proposed to tackle those limitations by extending SUs with appropriate interference-aware observability channels (e.g. the Cycle Contention Stack (CCS) [9], and the Request Duration Counter (RDC) [2]) and interference-aware controllability means (e.g. the Maximum-Contention Control Unit (MCCU) [2]). The CCS has been tested to some extent with a tailored implementation down to RTL for a LEON3 multicore, whereas the RDC and the MCCU have been tested only in performance simulators. Moreover, they have never been integrated synergistically. Thus, there is still a gap to be covered to integrate those technologies in commercial MPSoCs.

This paper tackles this gap by designing, implementing and evaluating the SafeSU, a SU particularly tailored for safety-relevant MPSoCs, to support the V&V of multicore timing interference concerns, and to provide means for the deployment of related safety measures. In particular, the SafeSU is a portable AMBA-compliant SU including the RDC, CCS and MCCU, which we integrate in two space-graded industrial MPSoCs, namely the RISC-V based Cobham Gaisler's NOEL-V MPSoC, and the SparcV8 based Cobham Gaisler's LEON3 MPSoC.

This work has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement EIC-FTI 869945.

II. Background & Related Work

Safety-related real-time systems must undergo strict V&V processes and include safety measures. This implies including appropriate observability and controllability means to manage timing interference in MPSoCs.

Hardware solutions, in terms of observability, are typically based on the use of Performance Monitoring Counters (PMC) to measure contention in multicore hardware resources [3], [13], [14]. Such solutions allow to reason about the maximum contention that the platform will experience. Regarding controllability, typically the most common approach is to isolate and partition shared resources, this is case of the cache for some embedded multicore processors [1], [8]. The problem with many of these solutions, is that either they require modifications to already existing FUBs, therefore difficulting it's implementation as it demands a costly functional verification process for the entire FUB, or do not provide enough insight with only end-to-end measurements.

Software solutions for increased observability are based on modeling and deriving the worst-case delays that tasks can suffer. This type of solutions usually use some sort of PMCs to ground their models on empirical measurements [7], [10]. For controllability, software solutions extend their models to enforce their bounds on maximum contention that tasks can experience by monitoring activity and stalling those contenders that exceed the amount of interference exercised [6], [11]. Software solutions often incur in high delays, overheads and overestimations, either in modeling solutions where the detail of the model directly relates to its computational cost, or in quota enforcement for which the OS or monitoring task has to run periodically generating overheads.

Overall, existing hardware solutions are either intrusive or harm performance, whereas software ones introduce non-negligible overheads and inherit the limitations of the limited hardware support.

III. SafeSU Baseline Components

Our SafeSU integrates the RDC [2], the CCS [9] and an enhanced version of the MCCU [2]. This section provides a necessarily brief presentation of the original components and their functionality, as devised by their authors (hence, without our MCCU enhancement).

A. RDC: Request Duration Counter

The RDC [2] includes a series of registers intended to monitor the duration in CPU clock cycles of different types of events observed in the interconnect. Such types of events can be any breakdown of the different types of events observable in the interconnect.

The RDC can operate in two different modes: verification and operation modes. In the former mode, the RDC [2] module monitors the duration in CPU clock cycles of each event e. If the maximum duration recorded for that event is exceeded, such maximum duration is updated. Therefore, the RDC can track the maximum duration observed for each request type. Those maximum values can be read and used for WCET estimation.

During operation mode, the RDC registers are set with reference values (e.g. typically those values recorded during verification mode), and used to track whether observed request durations exceed values used typically during verification. If any of such values is exceeded, an interrupt is raised, since there is some risk that the WCET used for task scheduling is not an upperbound to actual multicore interference. This functionality can also be used, with appropriate (high) initial values, that requests do not stall the interconnect for too long, thus behaving as a form of watchdog that prevents too long requests.

B. CCS: Cycle Contention Stack

The CCS [9] provides, for each core, how many cycles its requests have been stalled in the interconnect by each other core, and how many cycles it has used the interconnect. This is achieved by monitoring what cores request access to the bus, and which one is granted access. Thus, if a core c_i requests access and is granted access, it counts those cycles as "use cycles", but if another core c_j is granted access instead, those cycles are counted as interference from c_j. Note that this is done for all cores requesting access to the bus, so a given cycle will be counted by as many cores as cores requesting access to the bus during that cycle.

This information is particularly valuable for design optimization, validation, and to implement safety measures. If the contention caused by one core on another is regarded as too high, they can be scheduled differently. Analogously, if the contention caused by one core on another is overly high, this can be detected during validation to take corrective actions if needed. Finally, safety measures can be implemented to manage overruns during operation. For instance, upon a deadline overrun, the CCS for the task missing its deadline allows determining how much each other core contributed to the contention experienced by the offended task, and use a degraded operation mode disabling offending tasks (if that is possible) or resorting to alternative precomputed task schedules where the offended and offending tasks do not run concurrently.

C. MCCU: Maximum-Contention Control Unit

The MCCU [2] allows allocating multicore timing interference quotas to each core (in CPU cycles), and monitoring such quotas so that, whenever a core exceeds its allocated quota, an interrupt is raised so that the hypervisor/OS can take corrective actions (e.g. safety measures).

The MCCU subtracts to the quota of the core granted access to the interconnect the value of the RDC for the corresponding transaction event as many times as contenders are indicated as sensitive to interference. For instance, $core_0$ quota can be set for interference on $core_1$ and $core_2$, but not on $core_3$, which may run a non-critical or non-real-time task. Thus, the quota is set to account the aggregated interference caused in $core_1$ and $core_2$ only. For instance, if the operation performed through the interconnect by the core granted access is a read operation, the RDC for read

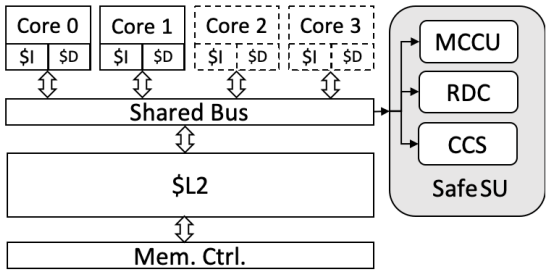

Fig. 1. High-level schematic of an SoC

operations will be subtracted from the quota as many times as contender cores are set as sensitive, thus accounting for the maximum possible contention the granted core can have caused on those other cores.

IV. SafeSU Realization

The SafeSU is a modular and scalable SU that can be connected to any on-chip shared interconnect (i.e. with multiple masters) and allows multicore interference observability and controllability (per interconnect master) building on the RDC, CCS and MCCU.

A. SafeSU Architecture

Figure 1 shows how our SafeSU is integrated into an example scalable SoC. Our SafeSU is attached to a single level of shared communication (e.g. shared bus), and monitors signals in the interconnect as well as in the elements connected to it (i.e. core and cache events in our example figure). As shown, the SafeSU element works as a standalone externally attached module, therefore freeing the integrator from costly re-verification of existing FUBs.

We have integrated our SafeSU in two SoCs for the space domain, a RISC-V SoC building on NOEL-V cores [5] and a SparcV8 SoC building on LEON3 cores [4], both by Cobham Gaisler, with both designs resembling Figure 1. Both designs use a shared AMBA AHB bus to connect cores to upper memory hierarchy levels (either a shared L2 cache or a memory controller).

B. RDC, CCS and MCCU Integration

Our integration of the RDC has been performed accounting for the latencies of two types of events: data and instruction cache misses in the core-local first level caches. We have deployed the complete set of functionalities of the RDC, thus allowing to measure the highest latency observed per event type in verification mode, and raising an interrupt in operation mode when the pre-programmed latency is exceeded.

The CCS has been integrated for the two SoCs aforementioned, which include up to 4 cores. Therefore, for an N core setup (e.g. 4 cores), the CCS requires N^2 registers (e.g. 16 registers for a 4-core setup) to track the contention that each one of the N cores can cause on each one of the other $N-1$ cores, and how many cycles each core spends using the bus.

The MCCU has been integrated in two different ways exploiting the fact that the SafeSU includes, apart from the RDC, also the CCS. Hence, we have performed the default integration of the MCCU where quotas are decremented based on the RDC values for each event type. However, we have also enhanced the MCCU to allow decreasing quotas by the actual latency caused on each contender, which is naturally measured by the CCS.

TABLE I
SUMMARY OF REGISTERS FOR THE PROPOSED SAFESU.

Component	Num. Registers			Size (bits)
	2 cores	4 cores	8 cores	
RDC	2	2	2	8
CCS	4	16	64	32
MCCU with RDC	2	4	8	32
MCCU with RDC and CCS	2	4	8	32
	2	4	8	8
Config register	1	1	1	32
Max total bytes	32	90	302	

Overall, the SafeSU, includes a number of registers and logic for the RDC, the CCS and MCCU. In our particular integration, the main cost relates to registers, which we summarize in Table I for the instantiation of the SafeSU for 2-core, 4-core and 8-core setups. Note that only the 2-core and 4-core setups have been realized, and registers for the 8-core setup have been estimated.

Note that apart from those registers, few configuration bits are needed to set the operation mode of the RDC, to reset the registers of any component, and the like. An additional 32-bit register suffices for that. For the sake of completeness, we also provide the total number of registers per core count considering the MCCU with RDC and CCS, which leads to a slightly higher number of registers required than the baseline MCCU with the RDC only. As shown, the cost is dominated by the CCS, which has quadratic – yet low – register requirements on the core count.

C. Interfacing

The SafeSU uses standard interfacing for AMBA Advanced High-performance Bus (AHB) interconnects, but it has been architected to be easily integrated with other bus protocols. This allows the SafeSU IP to be easily embedded into any design that has AMBA buses, such as Cobham Gaisler's MPSoCs targetting the space domain. In our implementation, the SafeSU is connected through an AHB interface to the MPSoC. This allows the SafeSU registers and counters to be written and read through usual load and store operations provided the adequate addresses are given.

V. Validation

This section presents some validation experiments for the SafeSU features, which has been implemented in SystemVerilog. We intend to release it as an open-source IP once commercial-level validation completes.

A. Experimental Setup

We validate the proposed module by synthesizing both multicores, namely RISC-V and SparcV8 SoC respectively, both by Cobham Gaisler, into a Xilinx Kintex UltraScale

TABLE II
SUMMARY OF MICROKERNELS

μkernel	Description
il1miss	Branches missing on L1 instruction cache
ldmiss	Loop of loads always missing on L1 data cache
ld2miss	Loop of loads always missing on L2 cache
st	Loop of store instructions

TABLE III
RDC MONITORED MAXIMUM REQUEST LATENCY IN DIFFERENT MICROKERNELS

Workload	il1miss	ldmiss	ld2miss	st	MAX
imiss	29	31	57	31	57
dmiss	0	45	63	0	63

KCU105 and Xilinx Kintex-7 KC705 evaluation kits respectively. Due to lack of space, we only present results for the SparcV8 SoC. The SparcV8 design is composed of 4 LEON3 cores [4], part of a space commercial platform, whose architecture is shown in Figure 1. In this evaluation we resort to bare-metal execution and leave OS and hypervisor integration as part of the future work.

We use a combination of benchmarks to validate SafeSU's functionality. We resort to the *cacheb* EEMBC Automotive benchmark, as well as custom made microkernels that stress specific FUBs of the processor. In Table II we summarize the names and description of each of those microkernels.

B. RDC Validation

We run our microkernels in Table II in isolation with the RDC in verification mode. The latencies observed for each of the two event types monitored are shown in Table III. As expected, the *ld2miss* microkernel leads to the highest latencies. Overall, this experiment provides some evidence of the correct operation of the RDC.

C. CCS and MCCU Validation

For space efficiency, we validate both the CCS and the MCCU with a single experiment. We use the *cacheb* EEMBC Automotive benchmark against an increasing number of *ld2miss* microkernel instances. Note that, since this SoC lacks an L2 cache, the bus is not released until memory transactions are served. Therefore, each bus transaction can potentially create large interference. Results are shown in Figure 2, where we see how the CCS effectively breaks down contention across contender cores in this setup. Note that, core pipelining allows overlapping some computation cycles with interference. Those cycles are counted as interference, and thus computation cycles apparently decrease with interference in the plot. Regarding absolute contention, we exercise control with the MCCU (enhanced using the CCS measurements to consume quotas) enforcing interference not to exceed 2x the isolation execution time, which the MCCU allows achieving, as shown in the figure.

D. Hardware Costs

We synthesize the SafeSU using the Vivado 2018 Toolchain and target the FPGA present in the Xilinx Ultra-

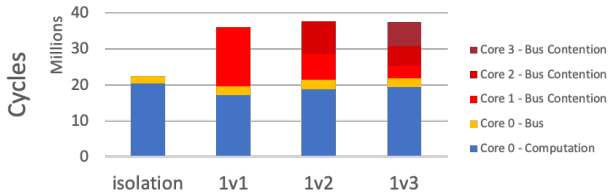

Fig. 2. Interference breakdown in the 4-core SparcV8 SoC.

Scale KCU105. The overall cost of our SafeSU implementation is around 5000 LUTs, whereas the entire SoC with 4 cores, but without L2 cache uses 192,400 LUTs. Overall, the SafeSU is a low-cost component representing just 2.6% of the entire SoC. Integrating the SafeSU into more complex SoCs would increase SafeSU costs negligibly – if at all – thus reducing its relative cost w.r.t. the SoC. Note that, for instance, L2 caches may easily account for close to half of the area of an SoC, which would roughly halve the SafeSU relative cost.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we present the SafeSU, an extended statistics unit particularly intended to provide observability and controllability means for multicore timing interference. The SafeSU successfully integrates some research solutions like the MCCU, RDC and CCS in commercial MPSoCs with low hardware overhead. Moreover, we show that, by integrating those components together, some synergies can be effectively exploited to increase functionalities, such as the proposed use of CCS values in the MCCU. Overall, the SafeSU is an effective and portable component easing the design of safety measures against multicore timing interference, as well as the verification and validation of the timing behavior of MPSoCs.

REFERENCES

[1] ARM. *ARM Cortex-A9 Technical Reference Manual r4p1*, 2016.
[2] J. Cardona et al. Maximum-contention control unit (mccu): Resource access count and contention time enforcement. In *DATE*, 2019.
[3] S. Chattopadhyay et al. A unified wcet analysis framework for multicore platforms. *ACM Trans. Embed. Comput. Syst.*, 13(4s), 2014.
[4] Cobham Gaisler. LEON 3 Processor. https://www.gaisler.com/index.php/products/processors/leon3.
[5] Cobham Gaisler. NOEL-V Processor. https://www.gaisler.com/index.php/products/processors/noel-v.
[6] D. Dasari et al. Response time analysis of COTS-Based multicores considering the contention on the shared memory bus. In *Int. Conf. on Trust, Security and Privacy in Computing and Communications*, 2011.
[7] M. Fernández et al. Assessing the suitability of the ngmp multi-core processor in the space domain. In *EMSOFT*, 2012.
[8] Cobham Gaisler. *NGMP - Data Sheet and User Manual*, 2011.
[9] J. Jalle et al. Contention-aware performance monitoring counter support for real-time MPSoCs. In *SIES*, 2016.
[10] H. Kim et al. Bounding memory interference delay in cots-based multi-core systems. In *RTAS*, 2014.
[11] R. Pellizzoni et al. Worst case delay analysis for memory interference in multicore systems. In *DATE*.
[12] J. Perez-Cerrolaza et al. Multi-core devices for safety-critical systems: A survey. *ACM Comput. Surv.*, 53(4), 2020.
[13] H. Shah et al. Measurement based wcet analysis for multi-core architectures. In *RTNS*, 2014.
[14] L. Zhao et al. Cachescouts: Fine-grain monitoring of shared caches in cmp platforms. In *PACT*, 2007.