Unboxing the Sand: on Deploying Safety Measures in the Programmable Logic of COTS MPSoCs

Sergi Alcaide, Guillem Cabo, Francisco Bas, Pedro Benedicte, Fabio Mazzocchetti, Francisco Cazorla, Jaume Abella

Unboxing the Sand: on Deploying Safety Measures in the Programmable Logic of COTS MPSoCs

Sergi Alcaide[†], Guillem Cabo[†], Francisco Bas[†,‡], Pedro Benedicte[†],
Fabio Mazzocchetti[†], Francisco J. Cazorla[†], Jaume Abella[†]
[†]Barcelona Supercomputing Center (BSC)
[‡]Universitat Politècnica de Catalunya (UPC)

***Abstract*—The lack of sufficient hardware support for functional safety precludes the full adoption of many Commercial Off-the-Shelf (COTS) MPSoCs in safety-related systems, such as those in the aerospace industry. Some recent MPSoCs come along with programmable logic (PL), primarily intended to offload some specific complex functions that can be much more efficiently implemented in hardware than in software, hence being such PL a kind-of-sandbox fully mastered by ASIC cores outside the PL.**

This paper proposes using PL in those COTS MPSoCs to deploy the support needed to implement safety measures efficiently to enable the use of those MPSoCs for systems needing high assurance levels. Hence, the goal is not mastering PL from the cores solely, but also allowing PL to provide monitoring (e.g. contention, diversity, watchdogs) and control (e.g. configuring QoS features) capabilities to enable the realization of a safety concept atop. The early work presented in this paper already provides specific monitoring, diversity, and controlling strategies to allow PL take over safety-related functionalities.

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

I. Introduction

Increased automation and autonomy in safety-related systems requires higher performance platforms able to execute those safety-critical tasks within tight time bounds. This can be achieved by using high-performance heterogeneous MultiProcessor Systems-on-Chip (MPSoCs) that include some form of accelerator (e.g. GPU, DSP, vector accelerator, and the like). For instance, platforms such as the NVIDIA Drive PX2 in the automotive domain [1], and the Xilinx Zynq UltraScale+ in the avionics domain [18] emerge as candidates to meet the corresponding performance goals.

Unfortunately, while those platforms provide the raw performance required, they challenge certification against safety standards due to their limited support to implement safety measures atop [10]. In particular, safety-related MPSoCs are generally expected to come along with native hardware support for independent watchdogs, diverse redundancy, error detection, etc. This is, for instance, the case for the Infineon AURIX processor family often used in the automotive domain [8], which on the other hand provides limited performance. Overall, end users face a conundrum between using platforms with appropriate hardware support to implement the safety measures needed at system level [9], [11], but insufficient performance, or using high-performance platforms lacking sufficient native hardware support to deliver mandatory safety measures.

As part of our recent work, we have devised and deployed a number of hardware components highly convenient to implement safety measures on top, as well as to improve testability, such as a multicore interference-aware statistics unit (SafeSU) [5], a module to enforce diversity across cores executing tasks redundantly (SafeDE) [2], and a programmable on-chip traffic injector to test timing and functional behavior during MPSoC validation and during operation (SafeTI) [12]. Those components are undergoing the final steps of their integration [7], [13] in commercial NOEL-V based MPSoCs for the space domain by Cobham Gaisler [6], and are offered as open source components [4]. However, those components rely on being integrated in MPSoCs during the design phases, hence with the ability to introduce some – yet limited – modifications related to the observability of some signals and control of some features.

Some MPSoCs, such as the Zynq UltraScale+ family, include some Programmable Logic (PL) as part of the SoC, typically intended to implement efficiently some functionalities, where the ASIC cores act as masters, and the PL as slave (e.g. working as an accelerator where cores offload some computation). In this context, the PL can be seen as a sandbox just responding to requests from the cores, where the latter truly exercise control over the MPSoC.

This paper contends that, enabling the use of high-performance MPSoCs for safety-related applications can be achieved by leveraging PL as a means to deploy hardware support to implement safety measures in COTS MPSoCs. In particular, we note that, if privileges (e.g. user mode, supervisor mode, etc.) are managed properly, functionalities in the PL can span beyond the sandbox by monitoring autonomously parts of the SoC and taking actions to control a subset of the MPSoC features. To illustrate this approach, in this work, and focusing on the Xilinx Zynq UltraScale+ MPSoC as a research vehicle, we show how safety-related hardware components can be deployed in the Zynq's PL to implement a number of safety measures such as (1) multicore interference monitoring building on the SafeSU [5], (2) support for diverse redundancy building on the SafeDE [2], and (3) support for device diagnostics building on the SafeTI [12], among other features.

The rest of the paper is organized as follows. Section II briefly introduces the Xilinx Zynq UltraScale+ MPSoC, as well as SafeSU, SafeDE and SafeTI. Section III presents the strategies being studied to allow the successful integration of the latter components (and some others to be developed) in the MPSoC. Finally, Section IV concludes this paper with a discussion on the forthcoming developments and opportunities emanating from this work.

II. Background

In this work, we analyze how to deploy safety-related hardware components in the PL of a MPSoC. Without loss of generality, we focus on a Xilinx Zynq UltraScale+ (ZUS for short) MPSoC. Hence, this section introduces the ZUS, as well as the already available safety-related components.

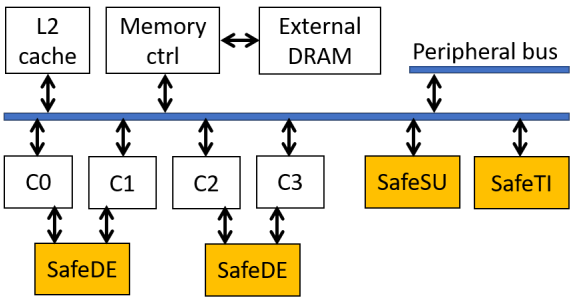

Fig. 1. Safety-related components as deployed in a NOEL-V MPSoC for the space domain.

A. Xilinx Zynq UltraScale+ MPSoC

The ZUS is a powerful MPSoC including a high-performance computing application cluster, referred to as APU, which includes 4 Arm Cortex A53 cores and a shared L2 cache. It also includes a real-time computing cluster, referred to as RPU, which includes 2 Arm Cortex R5 cores. Other computing elements such a GPU and PL are also present in the ZUS. Multiple memories and memory controllers are included in the ZUS, such as a DDR controller, an On-Chip Memory (OCM), and controllers to access flash memories. Multiple peripherals such as PCIe and Ethernet ports are also included. All those components are connected by means of a distributed network, so that traffic across different components can be fully segregated. For instance, independent routes exist from the APU to the DDR controller, from the PL to the OCM, and from the GPU to the DDR controller.

ZUS' interconnect builds on Arm components such as the CoreLink CCI-400 Cache Coherent Interconnect, and the CoreLink NIC-400 Network Interconnect, which include further Arm components, all of them implementing multiple and flexible QoS features, as shown in [14].

B. Hardware Components Supporting Safety Features

The work in this paper focuses initially in three already existing components supporting safety features, although we plan to develop and deploy additional ones. Those components, whose existing deployment in a commercial MPSoC is illustrated in Figure 1, are as follows:

SafeSU. The SafeSU statistics unit [5] includes observability and controllability channels to master multicore interference in MPSoCs. In particular, it collects statistics about how many cycles each master is delayed by each other master in an AMBA Advanced High-performance Bus (AHB) interface. Such information is particularly useful to diagnose timing overruns during operation and to optimize application deployment so that multicore interference is kept low.

The SafeSU also includes a multicore interference quota mechanism so that, if the observed interference caused by one master on another exceeds a user-programmed quota, an interrupt is raised. This allows limiting interference during operation.

Finally, the SafeSU includes specific logic to measure the highest latency experienced by a request in the AHB interface. Such information is collected per request type (e.g., read/write, burst/no-burst, etc.) and allows collecting maximum latencies used for Worst-Case Execution Time estimation, and also allows monitoring during operation whether latencies exceed a user-programmed threshold, which could be used as a form of watchdog.

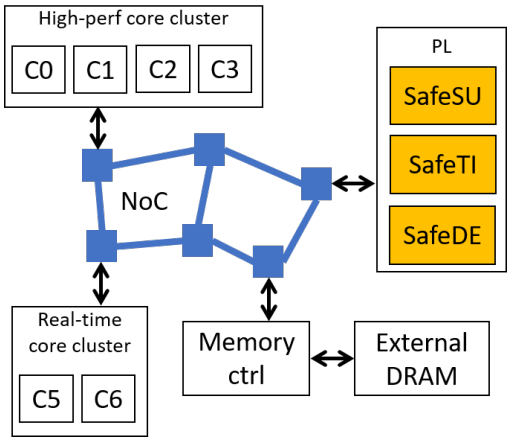

Fig. 2. Safety-related components deployed as proposed in this work in a Zynq UltraScale+ MPSoC.

SafeDE. The SafeDE is a module intended to enforce the staggered execution of a given task running redundantly in two cores. This feature is particularly useful to achieve some form of lockstepping (i.e. efficient diverse redundancy) in processors lacking it natively or, at least, lacking it for its highest performance cores. SafeDE collects the number of instructions executed by the cores running the redundant task, and whenever the advantage of the head core falls below a given threshold w.r.t. the trail core, SafeDE stalls the trail core for a while until the staggering is large enough.

SafeTI. The SafeTI is a sophisticated and programmable traffic injector able to inject specific traffic patterns whose elements are read and write operations, with parameterizable data transmitted, with user programmable source/destination, with a burst/no-burst parameter, with independent and programmable stalls between transactions, and with capability to store multiple independent or overlapping traffic patterns that may be used under different circumstances. As explained before, SafeTI is particularly adequate to test the platform during validation, as well as to test functionality and timing during operation.

While the ZUS MPSoC also includes its Xilinx AXI Traffic Generator (ATG) [17] in the PL, such IP is less flexible than SafeTI, and comes along a restrictive license, hence precluding its use in other platforms. Instead, SafeTI is provided under a highly-permissive open source license [12].

III. Deploying Safety Measures in the PL of the ZUS

Deploying hardware support to implement safety measures in a full-custom design, either deployed as an ASIC or as an FPGA product, provides flexibility to find the most efficient solutions. For instance, in the case of SafeDE, cores can be made to export some signals to let SafeDE easily monitor their progress and stall the trail core whenever needed. Analogously, SafeSU and SafeTI can be attached to any interconnect directly, hence achieving full observability of the on-chip traffic. However, if those components are deployed in the PL of a COTS MPSoC, observability and controllability channels are limited and cannot be changed. Hence, the challenge tackled in this work consists on how to deploy such hardware support

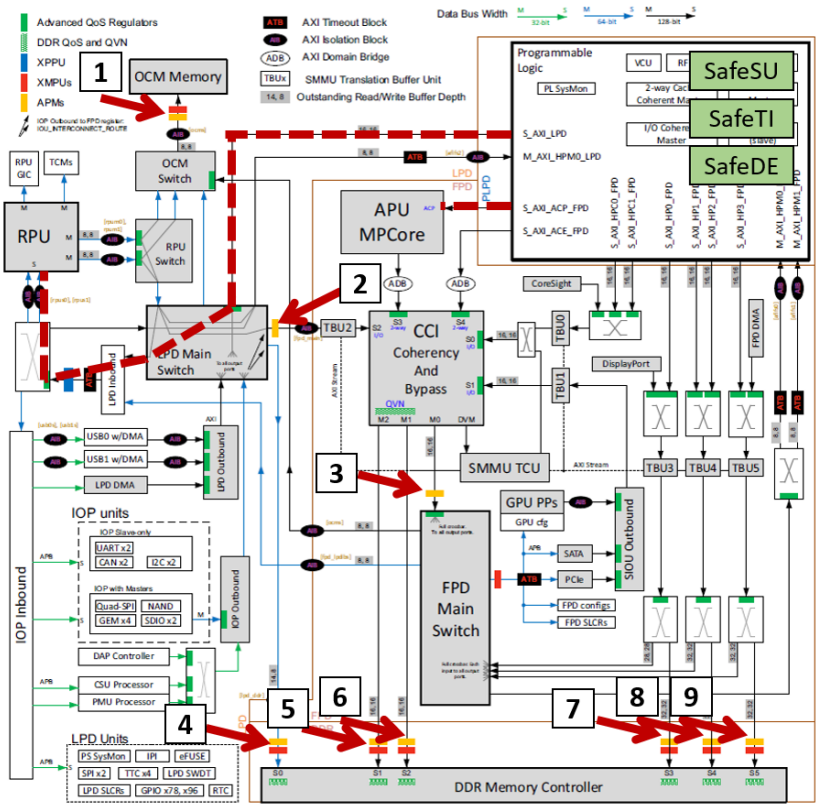

Fig. 3. Schematic of the ZUS MPSoC (baseline picture taken from [18]).

effectively in the PL, as illustrated in Figure 2 for the ZUS MPSoC.

A. Monitoring and Controlling Multicore Interference with SafeSU

Interference monitoring. The first relevant characteristic of the deployment of the SafeSU in the ZUS is that the ZUS has a distributed interconnect (see Figure 3), hence meaning that traffic is distributed rather than centralized in a single interconnect. This fact differs from previous deployments of the SafeSU, which built upon centralized interconnects (e.g. a bus) [15]. The second relevant characteristic is that the SafeSU cannot directly manage the AMBA AXI signals of the NoC, as it did with the AMBA AHB signals in its previous deployments. Overall, the SafeSU needs to collect NoC traffic information from remote locations and without direct access to protocol signals.

In order to properly integrate the SafeSU in the PL of the ZUS, we note that ZUS interconnects include AXI performance monitors (APM) [16], which gather transaction metrics such as the following:

- Read and write transaction counts.
- Read and write byte counts.
- Read and write latencies.
- Number of idle cycles caused by masters and slaves.
- Counts on some additional AXI-related protocol signals.

Moreover, APMs exist in different locations of the MPSoC, such as the interfaces near the DDR, the OCM, and the main switches connecting the APU and RPU computing units. They are shown as yellow squares in the ZUS schematic in Figure 3. They have been further indicated with thick dark red arrows and numbered. APM-1 monitors the OCM (aka as scratchpad memory). APM-2 monitors the traffic arriving from the Low Power Domain (LPD), where real-time cores are located, to the Full Power Domain (FPD). APM-3 monitors the traffic from the FPD, including APU cores, some of the PL ports and the GPU among others, to the LPD. APMs 4 to 9 monitor DRAM traffic for each of its different ports connected to the LPD, APU cores, PL, etc. In Figure 3, we have also included the SafeSU, SafeTI, and SafeDE in the PL for clarity. Hence, our current work focuses on interfacing APMs, as well as core-related memory access counters, to use their information to infer, either deterministically or statistically, how much interference each computing component has caused on each other. Moreover, the SafeSU needs to be extended to further break down interference across main locations to further ease diagnostics in case of deadline overruns.

Interference control. The SafeSU includes interference quota monitoring capabilities, and, upon a quota violation, it raises an interrupt. We note that the ZUS includes a wide variety of QoS knobs in its NoC interfaces, hence allowing to prioritize traffic based on its type and/or source. While not originally developed as part of the SafeSU, part of our work focuses on how to interface those QoS knobs so that the SafeSU can control them to limit specific interference channels whenever needed. This naturally needs being done in close collaboration with the Real-Time Operating System (RTOS), which should instruct the SafeSU on what knobs to set and how under quota violation scenarios, and must grant the SafeSU with appropriate privileges to change such configuration settings. Alternatively, the SafeSU can raise interrupts and be the RTOS the one in charge of configuring the QoS knobs as needed.

B. Enforcing Diverse Redundancy with SafeDE

Originally, the SafeDE has direct access to the instruction counts of the cores executing a task redundantly, so that it

can determine the staggering among them. SafeDE has also access to the stall signal of one of the pipeline stages of the trail core so that it can stop it almost immediately whenever the staggering is too low (e.g. below few cycles) [2].

In the context of the ZUS, the SafeDE can neither snoop instruction count registers nor control pipeline stall signals of the cores. Hence, alternatives are under consideration. Regarding instruction counts, we aim at, in cooperation with the RTOS, having means to read instruction counts from the *virtually lockstepped* cores as software would do. Note that paths to reach the cores in the RPU and APU from the PL exist in the platform. They have been indicated with dashed thick dark red lines in Figure 3. Hence, if the performance monitoring counters of interest are mapped into readable address spaces from the outside, and the RTOS programs privileges properly, SafeDE could read those counters from the PL and take an action whenever needed to preserve the staggering. Regarding stalling the trail core whenever needed, multiple alternatives are being considered such as:

- Modifying QoS knobs in the interconnect to favor the head core at the expense of slowing down the trail one. However, this is only effective if the task being run misses in local caches and accesses those interconnects.
- Issuing specific interrupts to the trail core so that, by programming them properly, the RTOS takes over for a short period intended to be enough to recover sufficient staggering between head and trail cores.

C. Diagnostics and Latency Measurements with SafeTI

The least impacted component due to being deployed in the PL of the ZUS is the SafeTI traffic injector since it issues transactions as programmed by the end user, regardless of whether the component is directly attached to the AMBA interface, or whether the interconnect is a bus or a NoC. Hence, integration of the SafeTI to generate traffic during operation for diagnostics purposes is not expected to bring major concerns.

We note, however, that the SafeSU may have difficulties to measure latencies of different request types by itself due to the lack of access to the AMBA interfaces, and due to the distributed nature of the ZUS' NoC. In this case, we plan to leverage the SafeTI to compensate that limitation since it can be programmed to produce specific traffic patterns triggering latencies that should allow building iteratively the latency to reach additional switches and components. Hence, by operating cooperatively, the SafeSU and the SafeTI will be able to measure the highest latencies experienced in most parts of the NoC.

D. Beyond Existing Components

Part of our ongoing work also includes the development of diversity monitors to complement SafeDE. A first version of those diversity monitors, referred to as SafeDM, has already been released [3]. Those monitors aim at measuring diversity across cores running redundant tasks, but not performing any control of their staggering. Such diversity is measured accessing pipeline information. However, it is still unclear how to measure diversity when pipelines are not visible. This is ongoing work for the ZUS which we expect to solve with as much precision as possible due to the limited observability of the cores.

Other components we intend to deploy in the PL of the ZUS relate to aliveness monitoring of different computing components with some form of watchdogs. Those will likely build on the instruction counts of the cores, and their NoC activity as primary sources of information related to aliveness.

Overall, the strategy is exploiting the (many) observability and controllability features of the COTS MPSoCs in general, and the ZUS in particular, with specific modules deployed in the PL to provide support for safety measures implementation.

E. Safety Considerations

By deploying safety measures in the PL of a COTS MPSoC, the set of assurance (integrity) levels that can be targeted are limited by the native safety support of the MPSoC itself. Hence, if the development process of the MPSoC does not adhere to the requirements of some assurance levels (e.g. DAL-A or DAL-B for avionics), then it is very unlikely that applications with safety requirements at those levels can be deployed on the MPSoC regardless the safety measures deployed in the PL. A sufficient assurance level must be attained at least for those parts of the MPSoC controlling the monitoring capabilities for fail-safe systems, and for those parts providing computing capabilities and monitoring capabilities for fail-operational systems. Else, external solutions may be required, such as the use of multiple MPSoCs with a sufficient degree of redundancy to meet the requirements of the highest assurance levels.

IV. Conclusions and Future Work

High-performance COTS MPSoCs needed for future aerospace systems lack sufficient native hardware support to implement efficiently many of the usual safety measures needed in those systems. We note, however, that some of those MPSoCs include a PL region which, despite generally intended to operate as a sandbox, can be used to deploy hardware components supporting safety features.

In this work, we review some existing such components and analyze how they could be deployed in the Xilinx Zynq UltraScale+ (ZUS) MPSoC, as representative example of high-performance COTS MPSoC, despite the gap existing between their original implementations and the observability and controllability channels available in the PL of the ZUS. In particular, we review the alternatives offered by the ZUS to monitor and control multicore interference with the SafeSU, to enforce diverse redundancy with the SafeDE, and to provide diagnostics with the SafeTI.

Our future work includes performing the integration of those hardware components in the ZUS to enable multiple safety measures in COTS MPSoCs, investigating additional hardware components that could be incorporated, and looking beyond the ZUS to consider even more powerful MPSoCs such as, for instance, the Xilinx VERSAL platform.

Acknowledgements

This work is part of the project PCI2020-112010, funded by MCIN/AEI/10.13039/501100011033 and the European Union "NextGenerationEU"/PRTR, and the European Union's Horizon 2020 Programme under project ECSEL Joint Undertaking (JU) under grant agreement No 877056. 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] NVIDIA DRIVE PX. Scalable supercomputer for autonomous driving. http://www.nvidia.com/object/drive-px.html.

[2] F. Bas et al. SafeDE: a flexible diversity enforcement hardware module for light-lockstepping. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2021.

[3] F. Bas et al. SafeDM: a hardware diversity monitor for redundant execution on non-lockstepped cores. In *2022 IEEE 25th Design, Automation and Test in Europe Conference (DATE)*, pages 1–6, 2022.

[4] BSC - CAOS. Safety-Related Hardware Components webpage. https://bsccaos.github.io.

[5] G. Cabo et al. SafeSU: an extended statistics unit for multicore timing interference. https://people.ac.upc.edu/jabella/ets21.pdf. In *IEEE European Test Symposium (ETS)*, 2021.

[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] Infineon. AURIX Multicore 32-bit Microcontroller Family to Meet Safety and Powertrain Requirements of Upcoming Vehicle Generations. http://www.infineon.com/cms/en/about-infineon/press/press-releases/2012/INFATV201205-040.html.

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

[10] J. Perez-Cerrolaza et al. Multi-core devices for safety-critical systems: A survey. *ACM Comput. Surv.*, 53(4), 2020.

[11] RTCA and EUROCAE. *DO-178B / ED-12B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[12] O. Sala et al. SafeTI: a hardware traffic injector for mpsoc functional and timing validation. In *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2021.

[13] SELENE Consortium. SELENE website, 2021. https://www.selene-project.eu/ (accessed Feb-2021).

[14] A. Serrano-Cases et al. Leveraging Hardware QoS to Control Contention in the Xilinx Zynq UltraScale+ MPSoC. In *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, 2021.

[15] G. Wessman et al. De-RISC: the first RISC-V space-grade platform for safety-critical systems. In *IEEE Space Computing Conference (SCC)*, 2021.

[16] XILINX. AXI Performance Monitor LogiCORE IP Product Guide. PG037 (v4.0). 2013.

[17] Xilinx. *AXI Traffic Generator v3.0. LogiCORE IP Product Guide*, 2019.

[18] XILINX. Zynq UltraScale+ Device. Technical Reference Manual. UG1085 (v2.1). 2019.