SafeLS: an Open Source Implementation of a Lockstep NOEL-V RISC-V Core

Marcel Sarraseca[†,‡], Sergi Alcaide[†], Francisco Fuentes[†,⋆], Juan Carlos Rodriguez[†],
Feng Chang[†], Ilham Lasfar[†], Ramon Canal[†,‡], Francisco J. Cazorla[†], Jaume Abella[†]
[†] Barcelona Supercomputing Center (BSC). Barcelona, Spain
[‡] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
[⋆] Universitat Autònoma de Barcelona (UAB), Barcelona, Spain

***Abstract*—Microcontrollers running safety-critical applications with high integrity requirements must provide appropriate safety measures to manage random hardware faults. For instance, automotive safety regulations (e.g., ISO26262) impose the use of diverse redundancy for items at the highest automotive safety integrity level (ASIL), ASIL-D. In the case of computing cores, this is realized with dual core lockstep (DCLS).**

The advent of the RISC-V ISA has made open source hardware gain popularity. However, there are few industrial open source SoCs meeting the requirements of safety-critical systems, and, to our knowledge, none of them provides lockstep cores.

This paper presents the realization of a RISC-V open source lockstep core based on Gaisler's NOEL-V core for the space domain, as well as its integration in the SELENE SoC that provides a complete microcontroller synthesizable on FPGA successfully assessed against space, automotive and railway safety-critical applications in the past.

I. Introduction

The development process of safety-critical systems must follow domain-specific guidelines and standards. In the case of the automotive domain, for instance, the main functional safety standard is ISO26262 [20]. In the context of ISO26262, four Automotive Safety Integrity Levels (ASIL) are defined, from ASIL-A (lowest, yet some, integrity requirements) to ASIL-D (highest integrity requirements). An additional level without safety requirements, Quality Managed (QM), is also defined.

ASIL-D is normally imposed for systems such as braking, steering and acceleration due to the severity of the consequences of a malfunction, the high exposure to a failure of those systems (i.e., they are used during a large fraction of the system operation), and the lack of controllability of the car upon a failure. ISO26262 imposes the use of safety measures to avoid failures caused due to random hardware faults, and, in the case of computing cores executing ASIL-D functionalities, the default solution is using Dual Core LockStep (DCLS). DCLS, or simply lockstep, consists of using two cores to run an instruction flow redundantly and transparently to the user, which only perceives the existence of one core. In automotive DCLS solutions, cores execute instructions redundantly with some staggering to guarantee that the state of both cores is different in electrical terms at any point in time to avoid the so-called Common Cause Failures (CCFs) [20]. CCFs are failures caused by a single fault affecting redundant elements analogously (e.g., a fault affecting the clock or power signals) so that, if their states are identical, they can experience identical errors that cannot be detected by means of comparison. Lockstep enforces different (diverse) states, and hence, even if a single fault can lead redundant cores to error, those errors will naturally differ and will be detectable. Lockstep cores have been realized in commercial designs and used in cars for a long time, e.g. Infineon AURIX microcontroller family [19], some STmicroelectronics products [39], and more recently NXP S32 platform products such as the S32G [28].

RISC-V Instruction Set Architecture (ISA) has gained popularity recently as a way to design and share open-source hardware IPs. Some chip and IP vendors, such as SiFive, have already delivered processors implementing lockstep, such as the SiFive E6-AD [36]. However, even those RISC-V-based designs are not open source.

This paper aims at closing this gap and delivering an open-source lockstep RISC-V realization, which we refer to as *Safe LockStep* or *SafeLS* for short. The SafeLS is based on a commercial open source RISC-V core for safety-critical applications, the Gaisler's NOEL-V core [10]. Moreover, we integrate it as part of an open source RISC-V based SoC already proven viable for automotive, space and railway applications, the SELENE SoC [14], [18]. In particular, the contributions of this work are as follows:

- We present the realization of the SafeLS, describing the technical decisions taken and the rationale behind them.
- We integrate the SafeLS as part of the SELENE SoC showing that it can operate transparently to the end user, analogously to non-lockstep cores.
- We assess performance and hardware costs showing that the SafeLS introduces completely negligible performance degradation – typically below 0.001% – and also low hardware costs with respect to the use of two cores redundantly without any lockstep mechanism to enforce staggering.

The SafeLS is currently fully functional, integrated into the SELENE SoC, and has also been extended with appropriate continuous integration support and documentation, so it will be shared as an open-source IP in the following weeks and, in any case, before the conference, along with other already-public SafeX components [4] in the webpage https://bsccaos.github.io/.

The remaining of this paper is organized as follows. Section II provides some background on redundancy and diversity. Section III presents SafeLS and its integration in the SELENE SoC. Section IV evaluates the performance and hardware cost of SafeLS. Section V reviews some related work. Section VI draws the main conclusions and future prospects of this work.

© 2023 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes,creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI 10.1109/IOLTS59296.2023.10224867

II. BACKGROUND

The development process of safety-critical systems imposes a series of steps for the design, implementation, verification and validation of the system so that the risk of software errors of any type, and systematic hardware errors causing a failure are deemed as residual, and hence, no safety measure is required during operation to manage their occurrence. However, radiation (e.g., particle strikes) and other types of electrical disturbance (e.g., crosstalk) can produce random hardware faults, which cannot be avoided and require appropriate safety measures to either prevent errors to occur or to correct them. Those safety measures are intended to make the risk of a failure due to random hardware errors become residual, as for any other type of failure.

Redundancy is the basis of many safety measures. For instance, data storage is often protected against errors by using Error Correction Codes (ECC) [9], such as, for instance, Single Error Correction Double Error Detection (SECDED) codes. Similarly, data transmission may rely on Cyclic Redundancy Checking (CRC) or on transmitting ECC-protected data to protect data exchange end-to-end against undetected errors that could become a failure otherwise. In the case of computing cores, the usual solution in safety-critical systems is using modular redundancy, such as Double Modular Redundancy (DMR) [13], [22], [25], [38] or Triple Modular Redundancy (TMR) [21], [23]. DMR is well-suited for detection and recovery through restoring a correct state (e.g., restart, or checkpoint recovery). TMR, instead, can be used to achieve fault tolerance by means of voting if faults lead to a single error, or to achieve error detection, as for DMR, if faults can lead to multiple errors as long as those differ. Note that, in both cases, if a fault leads to identical errors in two cores, those could not be detected by means of comparison in the case of DMR, or would lead to choosing as good the erroneous result in the case of TMR. As explained before, those failures caused by a single fault despite redundancy are referred to as CCFs.

Lockstep mechanisms are the usual solution to avoid CCFs. However, such lockstepping can be applied at a variety of granularities, each one with its pros and cons [17]. The granularity depends on the particular *Sphere of Replication* (SoR) used. The SoR determines the scope at which errors are detected. Any error occurring within the SoR can only be detected when propagated beyond the SoR.

DCLS sets the SoR at the core level, meaning that duplication and comparison occur at that scope. This is the SoR used in our work and illustrated in Figure 1. As shown, in this case, input that should be sent to one core is sent to both redundant cores, and their outputs are compared so that a single instance of those outputs is effectively sent. This specific SoR, DCLS, has been realized in multiple commercial products, as indicated before [19], [39].

The SoR can be set at a finer granularity (e.g., core stage level) [27], [38], which has some pros and cons. The pros are that detection occurs much earlier than in the case of DCLS, where an error may take longer to be detected since erroneous data may potentially spend millions of cycles in the core before crossing its boundaries [16]. The main cons relate to the high degree of intrusiveness of this approach if applied to existing cores, since all their pipeline stages need to be modified, and critical paths are likely to be affected. Also, it may detect errors that would be masked at coarser granularities (e.g., values never used, or not altering the result anyway)

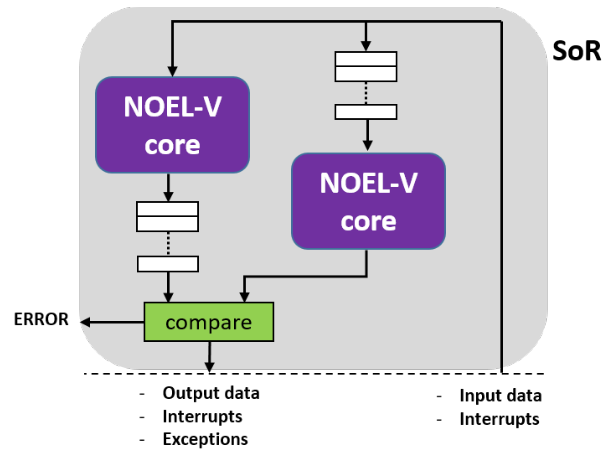

Fig. 1. High-level schematic of DCLS based on Gaisler's NOEL-V core.

One could also use a much coarser SoR by, for instance, using redundant servers [7]. This approach is onerous in terms of procurement costs and brings reliability concerns due to the existence of multiple physical devices, and the error exposure of the physical interconnects and interfaces across redundant servers. On the other hand, it provides the highest degree of independence, especially if those servers are different (e.g., different hardware, different ISA, etc.). This solution may be the preferred one for domains such as avionics, but DCLS, instead, is the one preferred choice for automotive, and the target of our work.

III. SAFELS DESIGN

This section presents SafeLS, our realization of the lockstep version of Gaisler's NOEL-V core, and its integration in the SELENE SoC.

A. SafeLS Realization

To realize the lockstep version of the NOEL-V core, we have reviewed its implementation and have determined that there are two main alternatives to generate the lockstep version of the core with limited intrusiveness:

- **FullCore**. Setting the SoR at the boundary of the full core, including the first level (L1) cache memories and memory management unit (MMU).
- **CachesExcluded**. Setting the SoR at the boundary of the core, but excluding L1 caches and the MMU, which can be protected with ECC.

The former choice, *FullCore*, has the advantage of not introducing logic in between the core and L1 caches, which would increase cache latency. Note that L1 cache latency, both for instructions and data, is typically critical for performance due to the fact that both caches are accessed every few cycles in the vast majority of the programs. Instead, increasing the latency for accesses beyond the L1 cache has, in general, a lower impact on performance since those occur much less frequently. The disadvantage of *FullCore* w.r.t. *CachesExcluded* relates to the fact that L1 caches and the MMU are replicated along with the computing core itself.

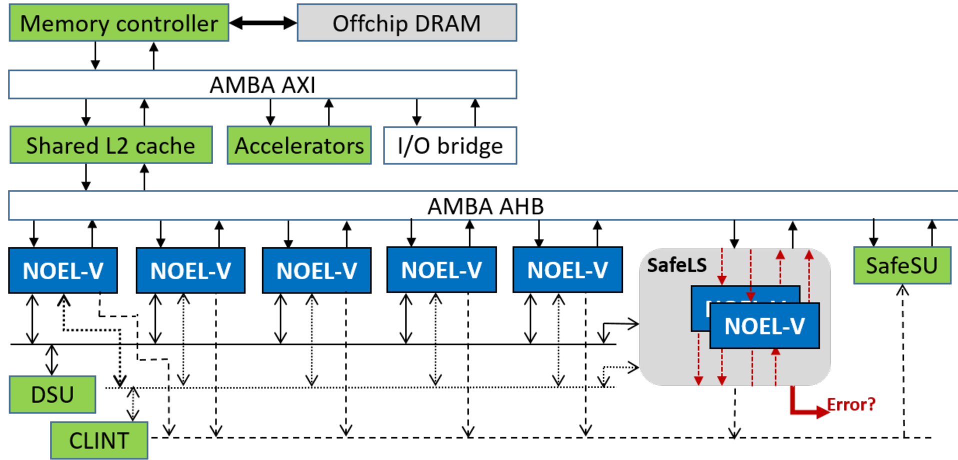

Fig. 2. SELENE SoC with SafeLS integrated for one core (now two in lockstep).

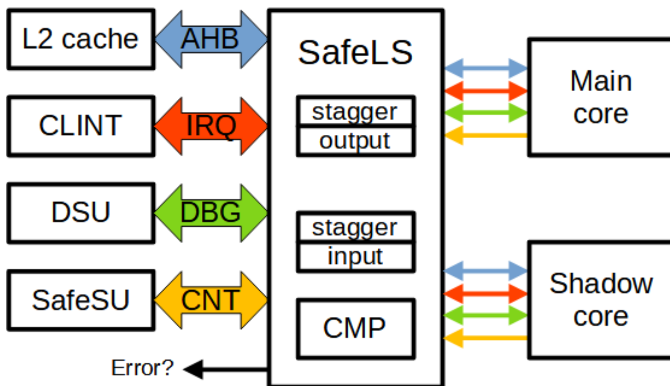

Fig. 3. Schematic of SafeLS as integrated into the SELENE SoC.

As expected, *CachesExcluded* pros and cons are complementary to those of *FullCore*, with L1 cache latencies impacted by as many cycles as used for staggering (e.g., typically 2-3 cycles), but reducing replication costs by not having to replicate L1 caches and the MMU.

In our particular realization of SafeLS, we have opted for the former, hence replicating caches but attempting to mitigate performance degradation. Part of our future work consists of realizing the latter option to offer different tradeoffs to users willing to integrate the SafeLS into their designs.

Given the *FullCore* SoR, the input and output signals for the SoR are as follows:

1) AHB-related inputs (`ahb_mst_in_type`, `ahb_slv_in_type`, `ahb_slv_out_vector`), and outputs (`ahb_mst_out_type`). These sets of signals connect the NOEL-V core with an AMBA Advanced High-performance Bus (AHB) interface, including incoming and outgoing data and addresses, and protocol interfacing signals. As shown in Figure 2, the AHB interface connects cores with a shared L2 cache.
2) Interrupt-related inputs (`nv_irq_in_type`), and outputs (`nv_irq_out_type`). These sets of signals serve the purpose of interfacing incoming interrupts, as well as those generated by the core itself.
3) Debug inputs (`nv_debug_in_type`), and outputs (`nv_debug_out_type`). These sets of signals include NOEL-V specific signals about its internal state and for its external control relevant for debugging purposes.
4) Tracing-related outputs (`nv_etrace_out_type`). These signals provide detailed information about the execution in the NOEL-V core intended to be collected by an external interface (e.g., a tracing unit) to record or output them.

To realize SafeLS, we have followed the scheme in Figure 1 for all input and output signals so that input signals are delivered immediately to the *main* core and with some programmable delay (N cycles) to the *shadow* cores. Analogously, outputs of the head core are delayed by N cycles, and compared against those of the shadow core as soon as they arrive, and delivered if the comparison is successful[1]. Upon a faulty comparison, an error signal is triggered. It is up to the SoC where SafeLS is integrated to manage the error.

B. SoC Integration

We have integrated SafeLS as part of the SELENE SoC, which already includes 6 NOEL-V cores. For our first integration, we have replaced one of the default NOEL-V cores with the SafeLS along with the two lockstep NOEL-V cores. The SELENE SoC, shown in Figure 2, includes an AMBA AHB to connect the cores to a shared L2 cache. A SafeSU module [8], which provides multicore interference observability and controllability, is also connected to the AHB. An AMBA AXI interface allows connecting the L2 cache to the memory controller, the I/O bridge, and some accelerators. The picture also includes Gaisler's Debug Support Unit (DSU) and the Core Local Interrupt Controller (CLINT).

For our integration, we note that the SELENE SoC does not use the core tracing signals. Hence, we drop their staggering management from SafeLS. Instead, the SELENE SoC includes a set of output signals directly fed into the SafeSU (`nv_counter_out_type`), so they need to be managed with duplication and staggering in the SafeLS, as for the other signals.

[1] During integration we found timing constraints that prevent delaying the outputs (our original design) and instead, the outputs from the main core are delivered directly. Further details can be found later in Section III-C.

As shown in Figure 2, all cores are connected to the AHB, the DSU, the CLINT and the SafeSU. External connections for the SafeLS are analogous to those of each other core, and duplication and staggering are managed internally in the SafeLS transparently for the SoC.

The specific schematic of the SafeLS once integrated is shown in Figure 3, where the main and shadow cores are both NOEL-V cores. As shown, the SafeLS exports an *error?* signal that is set upon a mismatch in the comparison between the outcomes of both cores.

C. Integration Limitations and Future Work

Our integration of the SafeLS in the SELENE SoC has two limitations that are part of our ongoing work. The first one relates to the management of the error signal, which currently is not exported to software. We are currently in the process of making it a core interrupt, propagating it to the CLINT, and capturing it at the software level whenever raised. We do not foresee difficulties in this task since we have already achieved such goal for the interrupts raised by the SafeSU.

The second limitation of the current SafeLS implementation relates to the strong timing constraints imposed by the AHB protocol implementation, which does not allow for delaying AHB outputs of the cores. Hence, we cannot stagger those outputs and compare them prior to delivering them to the AHB interface since doing so leads to a platform crash. Instead, signals from the main core are delivered immediately to preserve original timings and staggering is used for comparison and error detection. However, upon an error detection, the potentially erroneous output of the master core has already been delivered for a few cycles (i.e., as many as the number of cycles used for staggering). This requires either means to stop the propagation of such signal or, instead, modifying the AHB interface module of the core so that some staggering is allowed and those signals can be delivered to the AHB interface only after successful comparison. We are currently investigating the viability of the latter since it would allow preserving the canonical implementation of DCLS.

IV. Evaluation

A. Evaluation Framework

The SafeLS has been integrated as part of the SELENE SoC and synthesized into a Xilinx Kintex UltraScale KCU105 evaluation kit.

For our evaluation, we use the TACLe benchmark suite [11], which consists of a set of benchmarks with varying characteristics, such as the size of their code, the duration of the execution, and their cache locality, among other characteristics. The target of the TACLe benchmarks is the evaluation of critical real-time embedded systems, such as those intended for the SELENE platform. Those benchmarks are self-contained since their input data is already included in the source code files. Therefore, no data needs to be read from the disk or anywhere else explicitly, and they can be run straightforwardly on bare-metal setups such as one used for our experiments

Such a bare-metal setup has been used with the goal of having strong controllability on the experiments so that we avoid uncontrollable sources of execution time variation that would challenge our analysis of the results otherwise.

The rest of this section provides performance results as well as some results on the hardware cost of SafeLS. While fault injection results would also be desirable, there is no obvious way to perform such fault injection without performing the physical design of the processor and injecting faults at electrical level. Injecting single faults in one of the cores at a higher abstraction level would not test lockstep solutions since those faults only need redundancy to be detected, even if no diversity exists. However, injecting faults relevant to CCFs requires a way to determine the electrical impact that a fault (e.g., in the clock network) would have on two cores with different state to inject the appropriate simultaneous logical faults in both cores, but this is only viable using an electrical simulator for the ASIC implementation of the design, which has not been yet produced, although we aim at generating it in the future. Radiation campaigns would also be possible even on the FPGA implementation, but there would be no realistic way to tell whether faults injected are relevant for CCFs or not. Hence, useful fault injection campaigns are not doable yet, and remain as future work to be conducted once the physical design for an ASIC is synthesized.

B. Performance Results

In general, lockstep execution is expected to cause some tiny performance degradation due to the introduced latency for staggering purposes that has an impact when entering or leaving the SoR. In our case, this would have an impact on L1 cache misses, which are the events traversing the SoR. However, due to the difficulties to introduce some delay in the AHB-related output signals, output signals are delivered immediately by the main core and comparison for error detection occurs in parallel. Hence, no delay is introduced in the delivery of the core outputs beyond L1 caches, and hence, we only expect non-lockstep-related performance variations.

We have run each TACLe benchmark 1,000 times in the original setup (baseline) and the SafeLS setup and collected execution times. Slowdowns using the SafeLS in terms of average and median execution times are reported in Table I. Results are rounded using two decimals for the percentages, so some sporadic cases with tiny variations in the range (-0.005%, 0.005%) appear as 0.00%.

First, we note that 46 out of the 51 benchmarks either experience no execution time variation, or it is negligible, so their variation is rounded to 0.00%. Out of the remaining 5 benchmarks, 2 of them (`adpcm_enc` and `huff_dec`) experience some small variation in some runs[2] that leads to tiny – yet visible – average execution time variation. However, when we compare the median for those two benchmarks, we see no variation at all. Hence, we consider them also as benchmarks without relevant variation.

Three other benchmarks, namely `anagram`, `huff_enc` and `susan`, experience different degrees of variation, and such variations are in the order of few thousands of cycles. To further contextualize such variation, Figure 4 shows the normalized execution time span for the baseline between $\mu - 2 \cdot \sigma$ and $\mu + 2 \cdot \sigma$, where μ stands for the average execution time and σ for the standard deviation. If we assume a Normal

[2]Such variation can be caused by an unfortunate DRAM refresh or a spurious Ethernet packet generating timing interference in the AXI bus or memory controller.

TABLE I
TACLE BENCHMARK PERFORMANCE RESULTS.

Benchmark	Average		Median		Average	Median
	Baseline	SafeLS	Baseline	SafeLS	Slowdown	Slowdown
adpcm_dec	2781.3	2781.2	2781	2781	0.00%	0.00%
adpcm_enc	2610.3	2610.1	2610	2610	-0.01%	0.00%
ammunition	81774773.2	81774772.0	81774773	81774775	0.00%	0.00%
anagram	2210278.7	2212262.4	2205501	2186305	0.09%	-0.87%
audiobeam	852056.6	852056.4	852056	852056	0.00%	0.00%
binarysearch	102.0	102.0	102	102	0.00%	0.00%
bitcount	5288.0	5288.0	5288	5288	0.00%	0.00%
bitonic	10605.0	10605.0	10605	10605	0.00%	0.00%
bsort	42221.0	42221.0	42221	42221	0.00%	0.00%
cjpeg_transupp	608984.0	608981.8	608984	608982	0.00%	0.00%
complex_updates	3617.0	3617.0	3617	3617	0.00%	0.00%
cosf	74897.1	74897.1	74897	74897	0.00%	0.00%
countnegative	2183.0	2183.0	2183	2183	0.00%	0.00%
cubic	3131397.7	3131397.8	3131397	3131397	0.00%	0.00%
deg2rad	32304.0	32304.0	32304	32304	0.00%	0.00%
dijkstra	28796202.7	28796186.9	28796202	28796191	0.00%	0.00%
epic	8943689.1	8943685.8	8943689	8943686	0.00%	0.00%
fac	262.0	262.0	262	262	0.00%	0.00%
fft	226849.5	226849.7	226850	226850	0.00%	0.00%
filterbank	10231901.5	10231905.6	10231874	10231874	0.00%	0.00%
fir2dim	6359.0	6359.0	6359	6359	0.00%	0.00%
fmref	1676951.4	1676952.1	1676951	1676951	0.00%	0.00%
g723_enc	281155.3	281155.4	281155	281155	0.00%	0.00%
gsm_dec	801039.5	801039.6	801039	801039	0.00%	0.00%
gsm_enc	2385299.5	2385302.5	2385299	2385298	0.00%	0.00%
h264_dec	20873.0	20873.0	20873	20873	0.00%	0.00%
huff_dec	105823.1	105845.5	105823	105823	0.02%	0.00%
huff_enc	173974757.3	177349199.9	173860540	175598319	1.94%	1.00%
iir	576.0	576.0	576	576	0.00%	0.00%
insertsort	561.0	561.0	561	561	0.00%	0.00%
isqrt	302296.0	302296.0	302296	302296	0.00%	0.00%
jfdctint	1411.0	1411.0	1411	1411	0.00%	0.00%
lms	575062.2	575062.2	575062	575062	0.00%	0.00%
ludcmp	8170.0	8170.0	8170	8170	0.00%	0.00%
matrix1	3443.0	3443.0	3443	3443	0.00%	0.00%
md5	5271591.1	5271591.9	5271592	5271592	0.00%	0.00%
minver	4398.0	4398.0	4398	4398	0.00%	0.00%
mpeg2	135541717.3	135541725.4	135541715	135541723	0.00%	0.00%
ndes	40734.1	40734.1	40734	40734	0.00%	0.00%
petrinet	1749.0	1749.0	1749	1749	0.00%	0.00%
pm	27534356.0	27534355.6	27534356	27534356	0.00%	0.00%
prime	334.0	334.0	334	334	0.00%	0.00%
quicksort	1226731.4	1226738.5	1226731	1226732	0.00%	0.00%
rad2deg	32263.0	32263.0	32263	32263	0.00%	0.00%
recursion	62.0	62.0	62	62	0.00%	0.00%
rijndael_dec	2499778.6	2499777.5	2499780	2499778	0.00%	0.00%
rijndael_enc	2310337.2	2310335.0	2310337	2310336	0.00%	0.00%
sha	842333.0	842337.0	842333	842333	0.00%	0.00%
st	362919.9	362919.9	362920	362920	0.00%	0.00%
statemate	38484.8	38484.8	38486	38486	0.00%	0.00%
susan	21877833.7	21889178.8	21872248	21873156	0.05%	0.00%

distribution, such span includes 95% of the execution time values. Such region is depicted in blue for the benchmarks in the plot. The internal black line traversing those regions corresponds to the average execution for SafeLS. As shown, it falls within the 95% central region in the three cases, hence showing that statistically it cannot be claimed that SafeLS execution times are different from those of the baseline case.

When analyzing the causes of such variation, we can only conclude that it occurs mostly due to DRAM refresh interference, which is particularly relevant for these benchmarks because they are the ones accessing DRAM memory more frequently, and hence, are more exposed to DRAM access interference.

All in all, we can conclude that SafeLS is neutral in terms of performance impact in the context of its integration in the SELENE SoC.

C. Hardware Overheads

We have measured the cost of SafeLS in the FPGA measuring the LUTs required. The SafeLS module requires 4,714 LUTs out of the 102,508 required by the group consisting of the SafeLS and the two redundant cores. Note that the cost

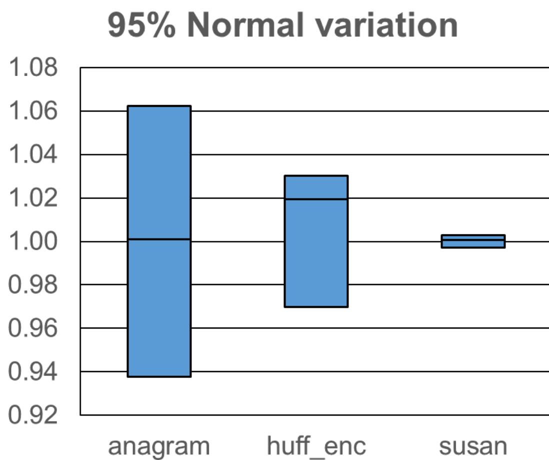

Fig. 4. 95% central Gaussian execution time distribution.

of one of the seven cores in the SoC ranges between 47,557 and 49,468 LUTs despite being identical, based on the Vivado Toolchain, which may need to introduce some differences to meet the FPGA frequency constraints (100 MHz). Therefore, the cost of SafeLS is below 10% of a NOEL-V core, and below 5% of the lockstep pair. If we consider the full SoC with the 7 cores, it requires 439,349 LUTs, so SafeLS requires less than 1.1% of all the LUTs in the SoC to make 2 of the 7 cores run in lockstep.

V. Related Work

Execution redundancy has been deeply studied in the literature. Solutions based on redundant multi-threading have been proposed to deal with transient faults [29], [32]. Redundancy across different cores has been shown to be also effective to deal with both, transient and permanent faults [13], [22], [25]. Some authors have also proposed solutions trading off redundancy and error detection capabilities [12], [24]. However, those works do not provide support against CCFs.

While previous works require some form of hardware support, other authors have investigated software-only solutions to achieve redundancy, also without explicit support for diversity [15], [26], [30], [33]–[35], [37].

Some of our past work includes a hardware module, SafeDE [6], or a software library counterpart [3], intended to enforce staggering across cores executing a task redundantly. While those solutions flexibly allow disabling or enabling diverse redundant execution, they run redundant user-visible processes, which, therefore, duplicate memory requirements and I/O interactions. The latter can change the functional behavior of the system, and hence, generally disallows the use of those solutions transparently, as opposed to DCLS, which we realize in this paper with an open source implementation in a commercial SoC for safety-critical systems.

Diverse redundancy in GPUs has also been the target of some software-only solutions for NVIDIA [2] and Intel GPUs [5], as well as of some works with hardware support [1].

Finally, some solutions to perform recovery with only dual core diverse redundancy have also been investigated in the literature [31].

VI. Conclusions and Future Work

Lockstep execution is mandatory in safety-critical systems with high integrity levels, such as ASIL-D automotive systems, due to the need for diverse redundant execution and reduced procurement costs. While lockstepping can be realized at different spheres of replication, DCLS has been proven a very effective tradeoff used in abundant automotive microcontrollers. However, to our knowledge, open-source implementations using commercial cores and industrially-viable SoCs have not been realized so far.

Our SafeLS design covers this gap by realizing a DCLS version of Gaisler's NOEL-V core for safety-critical systems. Moreover, we integrate SafeLS in a Gaisler technology-based open source SoC (the SELENE SoC), and show that its integration is highly efficient, incurs tiny hardware costs compared to the pure core replication (without lockstep support), and causes insignificant performance degradation. We will release SafeLS in a public repository in the next few weeks prior to the conference when a more exhaustive test campaign is complete, and aim at having the error signal properly driven to the interrupt controller.

As explained before, part of our future work consists of tailoring the AHB interface module to allow staggering AHB signals produced by the core so that comparison is possible prior to deliver the core outputs. This will make easier error management. Once this solution is in place, we plan to generate the physical design and perform fault injection of faults relevant for CCFs, such as faults in the clock and power grid networks.

Acknowledgements

This work is part of the European Union's Horizon 2020 Programme under project KDT Joint Undertaking (JU) under grant agreement No 101112274 (ISOLDE). 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] S. Alcaide et al. High-integrity gpu designs for critical real-time automotive systems. In *DATE*, 2019.
[2] S. Alcaide et al. Software-only Diverse Redundancy on GPUs for Autonomous Driving Platforms. In *IOLTS*, 2019.
[3] S. Alcaide et al. Software-only based diverse redundancy for asil-d automotive applications on embedded hpc platforms. In *DFT*, 2020.
[4] S. Alcaide et al. SafeX: Open source hardware and software components for safety-critical systems. In *2022 Forum on Specification and Design Languages (FDL)*, pages 1–4, 2022.
[5] N. Andriotis et al. A software-only approach to enable diverse redundancy on Intel GPUs for safety-related kernels. In *SAC*, 2023.
[6] F. Bas et al. SafeDE: a flexible diversity enforcement hardware module for light-lockstepping. In *IOLTS*, 2021.
[7] D. Bernick et al. Nonstop/spl reg/ advanced architecture. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 12–21, 2005.
[8] G. Cabo et al. Safesu: an extended statistics unit for multicore timing interference. In *2021 IEEE European Test Symposium (ETS)*, pages 1–4, 2021.
[9] C.L. Chen and M.Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state of the art review. *IBM Journal of Research and Development*, 28(2):124–134, 1984.
[10] Cobham Gaisler. NOEL-V Processor. https://gaisler.com/index.php/products/processors/noel-v, 2012.
[11] H. Falk et al. TACLeBench: A benchmark collection to support worst-case execution time research. In *WCET Workshop*, 2016.
[12] J. Fu et al. On-demand thread-level fault detection in a concurrent programming environment. In *SAMOS*, 2013.

[13] M Gomaa et al. Transient-fault recovery for chip multiprocessors. In *ISCA*, 2003.

[14] H2020 SELENE project consortium. SELENE hardware platform, 2021. https://gitlab.com/selene-riscv-platform/selene-hardware (accessed Mar-2023).

[15] F. Haas et al. Fault-tolerant execution on cots multi-core processors with hardware transactional memory support. In *ARCS*, 2017.

[16] C. Hernandez and J. Abella. LiVe: Timely Error Detection in Light-Lockstep Safety Critical Systems. In *DAC*, 2014.

[17] Carles Hernandez and Jaume Abella. Timely Error Detection for Effective Recovery in Light-Lockstep Automotive Systems. *IEEE TCAD*, 2015.

[18] C. Hernàndez et al. Selene: Self-monitored dependable platform for high-performance safety-critical systems. In *DSD*, 2020.

[19] Infineon. AURIX Multicore 32-bit Microcontroller Family to Meet Safety and Powertrain Requirements of Upcoming Vehicle Generations, 2012.

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

[21] X. Iturbe et al. Addressing Functional Safety Challenges in Autonomous Vehicles with the Arm Triple Core Lock-Step (TCLS) Architecture. *IEEE Design and Test*, 2018.

[22] C. LaFrieda et al. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *DSN*, 2007.

[23] R.E. Lyons and W. Vanderkulk. The use of triple modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, 1962.

[24] B. H. Meyer et al. Cost-effective safety and fault localization using distributed temporal redundancy. In *CASES*, 2011.

[25] S. S. Mukherjee et al. Detailed design and evaluation of redundant multithreading alternatives. In *ISCA*, 2002.

[26] H. Mushtaq et al. Efficient software-based fault tolerance approach on multicore platforms. In *DATE*, 2013.

[27] P.R. Nikiema et al. Design with low complexity fine-grained dual core lock-step (DCLS) RISC-V processors. In *SELSE*, 2023.

[28] NXP. *S32G2 Reference Manual, Rev. 7*, 2023.

[29] S. K. Reinhardt et al. Transient fault detection via simultaneous multithreading. In *ISCA*, 2000.

[30] G. A. Reis et al. SWIFT: Software implemented fault tolerance. In *CGO*, 2005.

[31] P. Reviriego et al. Diverse double modular redundancy: A new direction for soft-error detection and correction. *IEEE Design Test*, 2013.

[32] E. Rotenberg. AR-SMT: a microarchitectural approach to fault tolerance in microprocessors. *FTC*, 1999.

[33] J. D. Scales et al. The design of a practical system for fault-tolerant virtual machines. *Operating Systems Review (ACM)*, 2010.

[34] A Shye et al. Using process-level redundancy to exploit multiple cores for transient fault tolerance. In *DSN*, 2007.

[35] A. Shye et al. PLR: A software approach to transient fault tolerance for multicore architectures. *IEEE Transactions on Dependable and Secure Computing*, 2009.

[36] SiFive. SiFive Automotive E6-A. https://www.sifive.com/cores/automotive-e6-a.

[37] H. So et al. Expert: Effective and flexible error protection by redundant multithreading. *DATE*, 2018.

[38] L. Spainhower and T.A. Gregg. IBM S/390 parallel enterprise server G5 fault tolerance: a historical perspective. *IBM Journal of Research and Development*, 43(5/6):863–873, 1999.

[39] STMicroelectronics. SPC570Sx - 32-bit Power Architecture MCU for Automotive Chassis and Safety Applications, 2018.