We would like to thank reviewers for their valuable comments, which we have addressed to the extent possible. Please, find below answers and indications for each comment on how it has been addressed. Note that modifications in the paper to address specific reviewers’ comments have been performed in blue to ease their review, and will be turned into black once the paper is accepted. Analogously, other modifications to address more general comments related to unclear descriptions, and to fix errors, have been applied in magenta. Simple grammar errors and typos have been corrected but not marked.

**Reviewer: 1**

*1) It may be worth adding SafeDE to the title, for example: "SafeDE: A Low-Cost Hardware Solution to Enforce Diverse Redundancy in Multicores"*

ANSWER: We have modified the title accordingly.

*2) Abstract:*

*- Please rephrase this sentence as risk not been unreasonable sounds weird "Failure risk must not be unreasonable in high-integrity systems, such as those in cars, satellites and aircraft."*

ANSWER: the term “unreasonable” has been inherited from the main automotive functional safety standard, ISO 26262. As a similar comment has been raised in the introduction, here we rephrase it as “must be tiny” (instead of “must not be unreasonable”), and leave clarifications for other parts outside the abstract.

*- Please rewrite "behavior in front of faults."*

ANSWER: rewritten as “behavior in the presence of faults.”

*3) Introduction:*

*- "avoiding the unreasonable risk" -> "avoiding the unacceptable risk"*

ANSWER: rewritten as “unacceptable risk (aka unreasonable risk in ISO26262 terminology)” in line with the previous discussion about the abstract.

*- "are most common solutions" -> "are most the common solutions"*

ANSWER: rewritten as “are the most common solutions”

*4) Section III:*

*- "until CritSec2 is activated" I think it should be "until CritSec is activated"*

ANSWER: It is “CritSec2” since lockstepping is only activated when the second core enters the critical section. If only the first core enters the critical section, SafeDE does not take any action, so no staggering is enforced.

*- The TMR implementation is only discussed theoretically but not implemented. It would be nice to evaluate the impact on performance as it seems that staggering would be proportional to the number of redundant copies.*

ANSWER: Due to the limited time to perform this revision of the paper, we are unable to implement and validate the implementation of TMR. Note that the evaluation is performed in a 2-core setup. We have already available a newer setup with 4 cores, but also some other improvements in the SoC with side effects on the programmability of SafeDE. Hence, moving to this setup, which is part of our plans for evaluation in some industrial case studies of the H2020 SELENE project (<https://www.selene-project.eu/>), will occur in the near future, but not in time for this revision.

In any case, we can confirm that reviewer’s expectations on performance degradation match our expectations. Generally, adding an additional redundant core should increase execution time by very few tens of cycles. However, effects such as initial processor state, resource sharing, and the like, may create some larger performance variations hiding the expected trend to some extent, as for the 2-core setup.

*- "unnecessary stalls occur," I think it should be "unnecessary stalls do not occur,"*

ANSWER: the reviewer is completely right. Text fixed.

*5) Section IV:*

*- I think it would be good to run more FI experiments to get a relevant number of failures (as most are masked you only get a few failing runs that may not be enough to capture all relevant effects).*

Simulations are slow and performed in a FPGA shared with other people using it for some projects, so we could only move from 2,000 injections per fault model (original submission) to 4,000 injections per fault model (results in the current version). Trends did not change, so results discussion did not change meaningfully.

*- In Figure 6 it would be better to put the % of variation relative to the execution time in isolation. The current label of the y-axis is confusing.*

ANSWER: This has now been fixed.

*- In Figure 6 it may be better to also include the % for the SW based diversity to see the overhead of SW based protection.*

ANSWER: Unfortunately, the software-only solution is still being integrated onto the used RISC-V SoC, and this may take some weeks since the Linux distribution is not yet fully stable due to issue with the Ethernet interface. Hence, the software only solution is right now only working on an ARM target (those are the results we refer to in the paper, which we cite from [3]), as well as on an x86 target where we ported it to test its suitability. Hence, we won’t be able to provide results by the deadline to resubmit this paper (28th of February).

In any case, based on our experience on those two platforms, the execution time overhead of the software-only solution is typically slightly above the staggering set, which is moderate due to the time required to collect information and potentially stall the trail core through the operating system. In both setups (an ARM multicore and an Intel i7 processor), the lowest staggering we could safely use (determined empirically) was 100 microseconds. Such overhead is highly stable and independent of the duration of the program being executed. Hence, whether it is affordable or not strongly depends on the duration of the program, e.g. if it takes 10ms, then it is just a 1% increase, if it takes 50 microseconds, then it is a 3X increase factor.

**Reviewer: 2**

(No actual changes requested)

**Reviewer: 3**

*However the usage and the monitoring of the critical sections is not clear enough.*

*Some parts should be improved to make them more clear and more accurate, i list here under some of them:*

ANSWER: We have done an end-to-end review of the manuscript to gain clarity and conciseness, and to improve language and style.

*- in the abstract, line 27-28: "can be made operate" should be rephrased*

ANSWER: Rephrased as “can operate”. Now the full sentence is “cores can operate in lockstep mode efficiently or run independent tasks”

*- in the introduction l. 42: CCFs should be better defined here (which type of faults SA, SEU, ...)*

ANSWER: CCFs relate to the effect (the failure), rather than to the source (the fault). Hence, many types of faults could lead to a CCF. For instance, transient faults due to radiation or crosstalk, or latent defects, could propagate to the clock or voltage networks of redundant cores, or to their memory interface, and hence produce a CCF. We have clarified this matter in the introduction.

*- p2, l.42-43: "being the state of trail core...before" should be rephrased*

ANSWER: Rephrased as “so that the state of the trail core matches that of the head core N cycles before”

*- p2 column 2: "Note that THstag must be set to be large enough so that the trail core cannot execute those many instructions during Tcheck." => Not clear should be rephrased*

ANSWER: The explanation has been expanded to make it more clear.

*- p3 l.10: CritSec1 and CritSec2 determine whether the head and trail cores respectively are executing the code region needing lockstep execution." => i don't really get the behaviour of the usage of these registers, does they contain the adresses of the beginning of the critical regions ? => should be better explained*

ANSWER: The explanation has been expanded for clarity. Both registers act as flags to indicate whether the respective core has entered the critical region or not. Now it reads as follows: “CritSec1 (CritSec2) is set by the head (trail) core when it enters the code region needing lockstep, and reset when leaving it. Hence, lockstepping must be enforced when CritSec1 and CritSec2 are both set, as this indicates that both cores are executing the code region needing lockstep execution.”

*- p3 column 2: "Also, Safe DE, ...regions" should be rephrased, not clear*

ANSWER: Rewritten to improve clarity. The new sentence reads now like this: “Also, SafeDE may not be used for parallel programs if the number of instructions of any thread may vary depending on the order in which they get a specific lock since this could make redundant threads execute a different number of instructions.”

*- p4 column 2 l.54: "SafeDe" => typo*

ANSWER: Fixed.

*- p5 column 2 l.60: "afected" => typo and word not very appropriate here*

ANSWER: Rephrased. Now it reads: “They set the value of the bit selected for injection to 0, […].” We used “bit selected for injection” instead of “afected bit”.

*- p6 l.25: "The possible outcomes considered are:" you should explain why you use these metrics, especially DUE since this metric is stuck to 0 in your experiments. Why don't you consider other metrics ?*

ANSWER: The metrics have been chosen based on the following paper, which we failed to cite originally. We have fixed this concern and this is clarified right before the definition of the outcome categories:

Kaliorakis, M., Gizopoulos, D., Canal, R., & Gonzalez, A. (2017). MeRLiN: Exploiting dynamic instruction behavior for fast and accurate microarchitecture level reliability assessment. Proceedings - International Symposium on Computer Architecture, Part F1286, 241–254. <https://doi.org/10.1145/3079856.3080225>

They identify the following categories: Masked, SDC, DUE, timeout, crash, and assert.

Out of those, we keep identical: Masked, DUE, timeout, and crash. Regarding SDC, we call it “identical memory SDC” since errors are assessed at the end of the execution comparing memory dumps. Note that, since we target CCFs, SDC occur only when both redundant executions produce identical erroneous outcomes given that, if they differ, errors will be detected, but if they are identical there will be a CCF. Assert category does not exist in our experiments since it is specific of the type of experiments of the authors of the MeRLiN paper, who use such hint in their simulator. Finally, we added two additional categories relevant for CCF detection based on how errors manifest.

Regarding DUE category, note that we do not assume any specific error detection feature other than comparing results at the end of the execution. Hence, we considered as DUE only those errors that would be detected by the operating system due to, for instance, trying to access memory addresses out of the pages of the process. Such errors would be detected whenever they occur. We did not experience any such error but, since they are feasible and such category existed in the paper defining the categories (MeRLiN), we decided to keep it for completeness.

Last but not least, note that in our former version we had two separate categories for “Software detected” and “memory SDC”, depending on whether errors were detected comparing the result of the program or their memory contents. We have merged these two categories into “software detected” since both correspond to detectable errors by software comparison of the results.

**Reviewer: 4**

*I suggest to review language and grammar.*

ANSWER: We have reviewed the paper for language, grammar, and style to improve the quality of the manuscript.

*page 1, line 25: SafeDE is introduced in the IOLTS paper, not in this one.*

*You should make it clear in the abstract and in the introduction*

ANSWER: We have further made clear that this paper presents and extended analysis and evaluation of SafeDE. Note that the introduction already had this statement: “ In particular, SafeDE, which we first introduced in [6] and extend in this work, implements…”, where [6] is the original IOLTS publication.