The RERI-Lite error logging framework

Michiel Koenderink[a], Bruno Forlin[a], Gerard Rauwerda[a,b], Marco Ottavi[a,c]
[a]University of Twente, The Netherlands, [b]Technolution B.V, The Netherlands, [c]University of Rome Tor Vergata, Italy
[a]{m.j.h.koenderink, b.endresforlin, m.ottavi}@utwente.nl, [b] gerard.rauwerda@technolution.nl

***Abstract*—In recent years, the adoption of RISC-V cores in advanced systems has grown significantly. These cores are employed in several areas, including environments with a higher risk of hardware errors, such as space. Critical systems must detect and resolve as many errors as possible to maintain reliable operation. Error detection, logging, analysis, and resolution keep systems operational while collecting important diagnostic information. The RISC-V organisation proposed a specification for formatting error information, known as RERI. However, the extensive and large nature of this format can be impractical where time and resources are scarce. Furthermore, no dedicated framework around this error logging format has been specified or introduced yet. This work builds on the initial RISC-V RERI specification by introducing an adapted version called "RERI-Lite." Developed primarily for research use in radiation beam experiments, this system addresses the needs of smaller-scale applications with high error rates. This paper compares RERI-Lite to the standard RERI format. It demonstrates how the lighter, more flexible design of RERI-Lite can improve performance in resource-constrained contexts. Finally, the philosophy behind the error logging framework is examined, illustrating how it fits into broader system reliability goals.**

***Index Terms*—RISC-V, RERI, RERI-Lite, Error logging**

I. Introduction

RISC-V, as an open-source Instruction Set Architecture (ISA), has drawn increasing industry attention for diverse application domains such as industrial automation, healthcare, and space [1], [2]. This growing adoption highlights the critical need to address hardware errors that jeopardize Reliability, Availability, and Security (RAS). In radiation-intense environments, such as those found in space, systems become more susceptible to Single-Event Upsets (SEUs) and other radiation-induced faults, which can disrupt normal operations or even cause mission failure [3]. While various mitigation strategies (e.g., error detection, redundancy) exist, it is impossible to fully prevent hardware errors from occurring.

To enhance RAS, a dedicated subsystem can monitor, store, analyse, and report hardware errors. Such an Error Logging System (ELS) enables targeted error handling and corrective action. This capability is also beneficial in radiation testing environments, where multiple errors may accumulate and affect a device's functionality [4]. The RISC-V foundation has proposed a hardware error logging format called the RAS error record register interface (RERI) [5]. This specification describes a system that augments RAS features in the SoC with a standard mechanism for reporting errors by means of a memory-mapped register interface to enable error reporting [6]. Although the specification is comprehensive and flexible enough to also be included in embedded systems, many fields of the standard format will have to be left partially unused or unimplemented. This could lead to many different implementations of RERI records in embedded systems, making their compatibility with other applications difficult. For example, if a system wants to integrate components from different manufacturers that made changes to the RERI format, it will have to know what information is (no longer) available. This requires custom compatibility patches to integrate components. Using multiple RERI variants will make the integration more complex and it is likely to lower the efficiency of the system. The lack of a general standard will especially decrease the performance of systems like the ELS, which require compatibility with as many components as possible. That is why a single simplified version, based on the standard RERI format, that can be used by embedded systems would be a good addition to the original specification.

Therefore, this research proposes such an alternative: the RERI-Lite format, which, together with a general subsystem design, aims to more efficiently log and resolve hardware errors within RISC-V system-on-chips. RERI-Lite offers a streamlined approach that balances resource usage and system responsiveness. It keeps compatibility with the standard RERI specification by integrating it into the standard RERI bank and reusing the original RERI fields and codes. The aim of the RERI-Lite format is to be able to use together with the standard RERI format in a single system. Here we will present preliminary implementation results in a real system, considering data throughput, area and memory overheads, as well as presenting design considerations for improved versions.

II. Background

A. Error taxonomy and detection

The standard taxonomy for error detection has three different categories [7]. The first category is Silent Data Corruption (SDC), which contains the errors that are not detected. The second category, called Detected Uncorrectable Error (DUE) has errors that are detected, but these errors can not be corrected. The third one is Corrected Error (CE), which are errors that are detected and they can be corrected as well.

Hardware errors can occur in many different locations, which will influence the severity of the error. A simple bit flip in a data memory could overwrite a completely unused or irrelevant data value, while that same bit flip in the instruction memory could cause the system to overwrite important data, repeat or skip instructions or even fully break down.

For systems to be able to recover from soft errors, Error Detection and Correction (EDAC) is used. There are many different methods and techniques to detect and correct errors. However, these techniques focus on certain types of errors and these techniques all have their own advantages and disadvantages. There is not a single best way to detect and correct every possible error type that exists. Systems require a combination of methods to protect their system against all possible error types. So a general system to log all errors would need to be compatible with as many of these detection units as possible.

B. RERI Standard

The RISC-V foundation has specified a format for a bank with error records that can be used to store information about errors [5]. This format is called RERI. This specification describes an error bank that can log information about errors. One RERI bank can store up to 63 different error records. The first 64 bytes are used to store general information about the error bank, such as the bank ID, the amount of records in the bank and an overview of the valid records inside the bank. After that, every 64 bytes will form an error record. Each error record is made up of 8 registers of 64 bits.

The standard RERI error records are relatively large and specified for 64-bit based systems. Systems might not always have a lot of memory available to store error information, so in these systems, the implementation of a standard RERI error bank would be difficult. Besides, many fields of the error record will often be unused by errors specific error types. If fully implemented, not only would this waste memory, it could also slow down any processes that use or interact with these records. For example, reading a RERI record will require multiple bus operations to retrieve the error information. Specially in the process of analysing and reporting error information on a system with high error rates, speed will be important. The longer it takes to handle critical errors, the more additional damage these errors can cause.

Although the standard RERI specification provides full flexibility in which fields are implemented, if each design selects a custom implementation with different features, designing for RERI-enabled systems would be difficult as each ELS would be fully custom, and each monitored element could present a different interface. Especially for systems such as an ELS, creating new variants that are deviating from the original standard are problematic. Not only does this require additional implementation work, these modifications and specializations will most likely decrease the performance of the systems as well. Therefore, a smaller standard dedicated to embedded systems is required.

III. RERI-LITE

This work proposes an adjusted version called RERI-Lite that can be seen in Figure 1. This format uses 4 32-bit registers, making it more easily compatible with both 32-bit and 64-bit based systems. The top register is always used and its fields contain information about the component ID (*cid*), the error code (*ec*), the error priority (*pri*) and if the record is valid (*v*). The remaining fields will indicate how the other three registers are used. The *eat* field specifies if the register is used. For example, ”111” means that all the registers are used, while ”001” means that only the timestamp register is used. The *tst*, *ait* and *rif* fields can specify what type of format is used in the other registers.

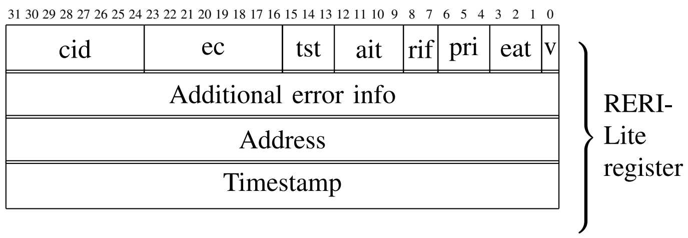

Fig. 1: The RERI-Lite error record format.

The *address* and *timestamp* registers store information about the address of the error and a time reference when the error occurred. Their information format depends on the code in the *ait* and *tst* fields in the top register. It works similar to the *ait* field used in the standard RERI specification [5]. The *additional error info* register is left as a customizable format for any information that is available about the error. Any custom format could be created based on a combination of the error code, the *rif* field and an additional format code that could be used inside the *additional error info* register itself.

This RERI-Lite format is more compressed and flexible than the standard RERI format. It allows error records to only use parts of the register that will actually contain useful information. At maximum, it uses only a fourth of the standard RERI error record size. The use of the record can be based both on the type of error and on the available system information. A system that has no form of timestamp can choose to ignore this register in general. Analysis or reporting units can use this flexibility to ignore unused parts of the record and to speed up their processes, resulting in better performance. In addition, systems with limited available resources can opt for RERI-Lite to save memory space.

IV. Error logging framework

To prove the concept of RERI-Lite, a framework was designed and built around the RERI-Lite error record bank as well. The RERI specification does not specify any framework around the RERI bank, so the system in Figure 2 was designed. This framework was designed with the intention to make it as generally applicable as possible.

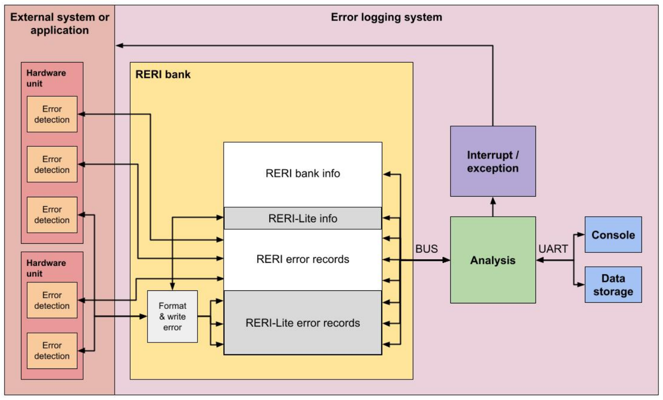

Fig. 2: A general overview of the ELS and the integration of RERI-Lite in the standard RERI format.

The entire process of the implemented framework can currently be split in four main stages: ”monitoring”, ”controlling”, ”analysing” and ”UART communication”. These stages can have several sub-processes and the required time for these processes can depend on the error type and other system variables. The different sub-processes can be seen in Table III.

The system uses the *format & write error* unit for the monitor stage to check all the error detection units inside the main system. Once an error signal is detected, it will gather the useful information and format it into a RERI-Lite error record. This unit will send the record to the RERI-Lite error bank. This error bank controls all error records. It will read and write the data from error records and reset records if the analysis is done with them. An analysis unit can then check the error bank for new error records and process them. This analysis is currently very simple and only transmits the error data to an external console by using a Universal Asynchronous Receiver-Transmitter (UART) communication link. This analysis can be extended in the future to allow for a more complicated analysis that will be able to provide useful error feedback to the core as well. The last stage is a UART unit. This unit can be connected to an external console and will make sure that the error data that is provided by the analysis will be transmitted according to the UART protocol.

V. Implementation Results

The framework design with RERI-Lite error records was synthesized and implemented targeting an Arty A7-35T board. The results of this implementation run are shown in Tables I and II. Since the ELS is not processing any signals, the implementation does not use DSP blocks.

Logic type		Amount of RERI-Lite records				
		1	4	8	16	32
Slice LUTS	Total	216	451	1929	3468	7226
	As logic	215	447	1921	3452	7194
	As memory	1	4	8	16	32
Slice registers	Total	254	534	2041	2638	6769
	As flip flop	248	528	2035	2632	6763
	As latch	6	6	6	6	6
Muxes	F7	0	0	50	20	122
	F8	0	0	8	10	8

TABLE I: A table with an overview of the slice logic usage for the implementation of the ELS with various amounts of RERI-Lite records. The LUTs implemented as memory are all shift registers.

Records	RERI			RERI-Lite		
Amount	Bytes	RAMB36	Utilised	Bytes	RAMB36	Utilised
1	64	4	8%	16	1	2%
4	256	16	32%	64	4	8%
8	512	32	64%	128	8	16%
16	1024	-	-	256	16	32%
32	2048	-	-	512	32	64%
64	4096	-	-	1024	-	-

TABLE II: A table with the (expected) amount of used RAMB36 blocks for the standard RERI and RERI-Lite implementations on an ARTY A7. The required RAMB36 resources of the standard RERI implementation are estimated.

As can be seen in Table II, the RERI-Lite format scales linearly and the format is expected to use only a fourth of the memory required by the standard RERI format. The ARTY A7 was able to implement 32 RERI-Lite error records, while the standard RERI format could only implement 8 records. So, systems that implement the RERI-Lite format can have a larger error buffer. An increased number of error records allows the system to better handle detected errors peaks. Especially in systems where a large fluctuation in the amount of errors is expected, it will be useful to create a large error buffer. So in systems with limited available memory, the RERI-Lite format can be used to increase the buffer size of the ELS, without the need of more memory.

Other advantages of reducing memory usage are the decrease in area and power. Since ELS’ memory needs to be protected, ECC increases both logic and memory requirements. Also, the larger the area used, the higher the chance that errors will occur in ELS, increasing the system’s vulnerability.

This area usage also applies to the power usage. Regardless of technology, the RERI-Lite format will always scale better than the standard RERI-format. The results of the RERI-Lite components give an indication of how well the standard RERI format would perform. Table I shows the number of on-chip components that are used by the ELS. The standard RERI format will require 4 times the memory of the RERI-Lite format. It is expected that the component and power usage of an ELS with standard RERI records will be comparable to 4 times as many RERI-Lite records. For example, when implementing a system with 4 or 8 standard RERI records, the system component and power usage would be comparable to implementing 16 or 32 RERI-Lite records. This means that the RERI-Lite format is likely to scale 4 times better than the standard RERI format if the power usage scales linearly.

As mentioned in the previous section, the ELS has four main stages. An overview of required clock cycles for the various processes is shown in Table III. The clock cycle data was gathered in a Vivado simulation. To be able to evaluate the RERI-Lite format, this overview also shows the expected amount of clock cycles that a standard RERI system would require.

Process	**Subfunction**	**Clock cycles**	
		RERI	**RERI-Lite**
Monitoring errors	Detect error signals	3 + N*	3 + N*
Control error record	Write valid record	33.5	9.5
	Reset record	33.5	9.5
Analyse errors	Read error record	149 - 150	14 - 42
	Check error records	3.5	3.5
	One bus transaction	9	9
UART communication	Create UART transmission	1	1
	Transmit UART frame	8681	8681
	Transmit full error record	607639	86806 - 190972

TABLE III: A table with the (expected) amount of clock cycles used to execute parts of the error logging operations. N represents the number of additional errors that were detected in the same clock cyle, since every additional error delays the timing by one clock cycle. The UART communication clock cycles assumes a 115200 baudrate and a 100 MHz clock frequency.

As can be seen in the overview of Table III, the RERI-Lite format is expected to perform much better than the standard RERI format in processes that require interaction with the error records. Since the error record format is only a fourth of the standard RERI format, it requires less write, read and bus operations to complete the important processes. Additionally, the RERI-Lite format is more flexible, so the system can ignore the unused parts of the error records. This can save a lot of unnecessary read operations and bus transactions. Reading an entire RERI record (assuming the use of a 32-bit width bus) will require 149 - 150 clock cycles to complete. In contrast, the RERI-Lite version will use a maximum of 42 clock cycles and the amount can even be lowered to 14 clock cycles if the error type uses only one of the registers. Even if other frameworks can increase the efficiency of the processes, RERI-Lite will always require less operations and will thus be relatively faster than the standard RERI format.

The same concept applies to the UART communication process. The standard RERI format will always have to transmit the information of the entire record, while the RERI-Lite format can ignore the unused parts and lower the required amount of UART frames. The reduced error record size is not only useful for faster error transmissions, but its reduced size can also prevent the loss of error data. In systems where a low baud rate is used or where an extremely high number of errors per minute is expected, the large standard RERI records can quickly overwhelm the UART connection and cause important error information to be lost. The RERI-Lite format will require a lot less UART frames on average, so the number of RERI-Lite errors that can be handled by the UART connection will be much higher.

VI. Conclusion

The main contribution of this work was to create a new error record format to store error information. This format is based on the RISC-V RERI architecture specification and is called RERI-Lite. It uses fields and encodings similar to the standard RERI format, but the key difference is that this new error record format is much smaller and more flexible. This allows smaller systems or applications that require fast error processing or report speed to increase their performance.

This work also designed and implemented a framework for the entire process of handling error information. It monitors error detection units and upon the detection of an error, it will format the available data about that error into the RERI-Lite format. Next, the system will process the error data and export it to an external console using the UART protocol. Future versions of this system could extend this analysis to also provide feedback to the main system. This will allow the ELS to provide a system recovery feature as well in the future.

References

[1] P. R. Nikiema, A. Palumbo, A. Aasma, L. Cassano, A. Kritikakou, A. Kulmala, J. Lukkarila, M. Ottavi, R. Psiakis, and M. Traiola, "Towards Dependable RISC-V Cores for Edge Computing Devices," in *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, Jul. 2023, pp. 1–7, iSSN: 1942-9401. [Online]. Available: https://ieeexplore.ieee.org/document/10224862/referencesreferences

[2] G. Furano, S. Di Mascio, A. Menicucci, and C. Monteleone, "A European Roadmap to Leverage RISC-V in Space Applications," in *2022 IEEE Aerospace Conference (AERO)*. Big Sky, MT, USA: IEEE, Mar. 2022, pp. 1–7. [Online]. Available: https://ieeexplore.ieee.org/document/9843361/

[3] "ECSS-Q-HB-60-02A – Techniques for radiation effects mitigation in ASICs and FPGAs handbook (1 September 2016) | European Cooperation for Space Standardization." [Online]. Available: https://ecss.nl/hbstms/ecss-q-hb-60-02a-techniques-for-radiation-effects-mitigation-in-asics-and-fpgas-handbook-1-september-2016-published/

[4] H. Quinn, "Challenges in Testing Complex Systems," *IEEE Transactions on Nuclear Science*, vol. 61, no. 2, pp. 766–786, Apr. 2014, conference Name: IEEE Transactions on Nuclear Science. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6786369

[5] "RISC-V RERI Architecture Specification," Sep. 2023. [Online]. Available: https://github.com/riscv-admin/ras-eri

[6] "RISC-V Technical Specifications," Jan. 2025, original-date: 2022-06-07. [Online]. Available: https://github.com/riscv-non-isa/riscv-ras-eri

[7] V. Sridharan, D. A. Liberty, and D. R. Kaeli, "A Taxonomy to Enable Error Recovery and Correction in Software," in *Workshop on Quality-Aware Design*, 2008.