Implementation and Reliability Evaluation of a ChaCha20 Stream Cipher Hardware Accelerator

Wesley Grignani, Khalil G Q Santana, Douglas Almeida dos Santos, Luigi Dilillo, Douglas R Melo

Title: Implementation and Reliability Evaluation of a ChaCha20 Stream Cipher Hardware Accelerator

Author(s): W. Grignani, K. G. Q. Santana D. A. Santos, L. Dilillo, D. R. Melo

Document version: Pre-print version (Final draft)

Please cite the original version:
W. Grignani, D. A. Santos, L. Dilillo, D. R. Melo, "Implementation and Reliability Evaluation of a ChaCha20 Stream Cipher Hardware Accelerator, "2024 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2024.

Implementation and Reliability Evaluation of a ChaCha20 Stream Cipher Hardware Accelerator

Wesley Grignani*, Khalil G. Q. Santana*, Douglas A. Santos†, Luigi Dilillo†, and Douglas R. Melo*
*LEDS, University of Vale do Itajaí, Itajaí, Brazil
†IES, University of Montpellier, CNRS, Montpellier, France
{wesley.grignani, khalil.santana}@edu.univali.br, {douglas.santos, luigi.dilillo}@umontpellier.fr, drm@univali.br

Abstract

Cryptography is fundamental to ensuring the security and privacy of space communications, enabling the reliable exchange of sensitive data between spacecraft, ground stations, and other components of space infrastructure. However, using encryption can be computationally costly, resulting in lower throughput or higher latencies. One strategy to mitigate the costs imposed by cryptography is using accelerators. In addition, these systems that operate in space are susceptible to faults due to adverse conditions and require the implementation of protection techniques to mitigate these faults and ensure correct operation. In this context, this paper presents a fault-tolerant and high-performance encryption accelerator for the ChaCha20 stream cipher. We present a low-cost optimized implementation and a fault-tolerant version using Hamming Error Correction Code (ECC) and Triple Modular Redundancy (TMR). Results show that the optimized and hardened versions accelerated the application by 23× and 17× compared to the software solution. The optimized solution can process 281.25 MB/s, presenting a higher throughput than state-of-the-art works that rely on FPGA implementation. The hardened solution can process 187.50 MB/s, leading to an overhead of 1.52× more Look-Up Tables (LUTs) and 1.03× more Flip-Flops (FFs) compared to the optimized solution of this work.

Index Terms

Cryptography, RFC 8439, Hardware Acceleration, Fault Tolerance.

I. Introduction

Encryption is widely used in various applications to enhance security and meet compliance requirements. However, implementing encryption can be challenging in scenarios with limited resources, such as embedded devices, due to potential overheads [1]. In space systems, data cryptography is vital for securing sensitive data transmissions and ensuring data privacy and security. The encryption protects satellites, subsystems, mission data, and the entire space-ground system [2].

Cryptography categorizes ciphers into several subdivisions, including symmetric, asymmetric, block-based, or stream-based. Stream cryptography involves generating a key stream from an initial key and combining it with cleartext to produce ciphertext [3].

To mitigate encryption/decryption overheads, cryptographic accelerators and lightweight encryption algorithms can be employed. The study [4] shows that ChaCha and ASCON offer faster performance and lower energy consumption than traditional AES ciphering. However, the same study concludes that ASCON has a higher cost in some of its functions, resulting in ChaCha being the best choice for most use cases.

ChaCha, a stream cipher proposed by Daniel J. Bernstein in 2008, is used in various software applications like WireGuard, OpenSSH, and TLS. It was standardized in RFC 8439, describing its key features and parameters [5]–[7]. Its adaptability to resource-constrained environments makes it well-suited for space systems.

Some works presented the development of hardware cryptographic accelerators for the ChaCha cipher. The work [8] introduced a scalable hardware implementation capable of achieving high throughput across different configurations for ChaCha8/12/20. Semenov [9] emphasized the potential of hardware acceleration, achieving a 3.61× speedup over software by optimizing logic resource usage even with certain operations, like adding internal states and XOR with cleartext, being performed in software. Similarly, the work [10] implemented the ChaCha20 cipher in FPGA, providing insights into resource utilization.

This work introduces a fault-tolerant ChaCha20 hardware accelerator aimed at high performance with low resource utilization. The accelerator was designed with a combination of spatial and information redundancies to improve the accelerator reliability, including the Hamming Error Correcting Code (ECC) and the Triple Modular Redundancy (TMR).

This work was supported in part by the Foundation for Support of Research and Innovation, Santa Catarina (FAPESC-2021TR001907), the Brazilian National Council for Scientific and Technological Development (CNPq - process 50794/2023-5), the Brazilian National Coordination of Superior Level Staff Improvement (CAPES/PROSUC), the EU project RADNEXT - Horizon 2020 (grant 101008126), and Project HARV in the framework of the action ”Accelerateur d’innovation” from the University of Montpellier.

II. Hardware Implementation

The accelerator was initially designed in a standard form (STD), utilizing a single instance of the `Qround` component and additional registers for storing intermediate results, which were periodically written back to the main state matrix register.

Subsequently, we developed an optimized design (OPT) featuring two sets of four instances of the `Qround` component, facilitating the computation of parallel column- and diagonal-wise rounds. These sets are interconnected via internal wiring to map each output of the first set to its corresponding input in the subsequent set. This configuration enables the computation of a full `DoubleRound` at once, simplifying the controller and enhancing throughput.

Considering the architecture design of standard and optimized solutions, we developed a fault-tolerant (FT) solution from the optimized version. We chose the optimized version because it uses fewer memory elements, which leads to fewer Single-Event Upsets (SEUs) in the design.

A. Hardware Architecture

The architecture of the hardened accelerator is illustrated in Fig. 1. In this optimized configuration, two sets of four elements compute the `Qround` function connected in series. The first set concurrently computes the four column-wise rounds using an instance of the `QuarterRound`. Subsequently, the outputs of these blocks are linked to the inputs of the subsequent set of `QuarterRound` blocks, which concurrently compute the diagonal-wise rounds. This approach eliminates the need for a write-back register to store intermediate results, as each clock cycle computes an entire `DoubleRound`.

The `Concat` component merges various cipher inputs, including the current block counter (`i_BLK_COUNT`), the key (`i_KEY`), and the `nonce` (`i_NONCE`). These inputs constitute $3/4$ of the initial state matrix, while the remaining 128 bits correspond to the constant "`expand 32-byte k`". The `MUX` switches between the initial state (`i_DATA_FROM_CONCAT`) and the state of the previous round (`i_DATA_FROM_LAST_ROUND`), feeding into the `RegPP` component.

The `QRound` function takes four 32 bit inputs as a 128 bit vector, which undergo various logic and arithmetic operations, including 4 additions, 4 XORs, and 4 rotations. The `DoubleRound` component encompasses two sets of four instances of the `QRound` component (totaling 8), arranged to execute all column-wise operations in parallel, followed by simultaneous computation of diagonal-wise rounds.

The `RoundCounter` signals through its output (`o_ALL_ROUNDS_FINISHED`) when all 20 rounds are done. The `SMAdder` module adds each of the 16 elements of the current state matrix (`RegPP`) to the initial state matrix (`Concat`), without carrying bits between matrix elements.

After all rounds, a bitwise XOR operation is performed between the cleartext and the generated One-Time Pad (OTP), generating the ciphertext in the output `o_CIPHER_TEXT`.

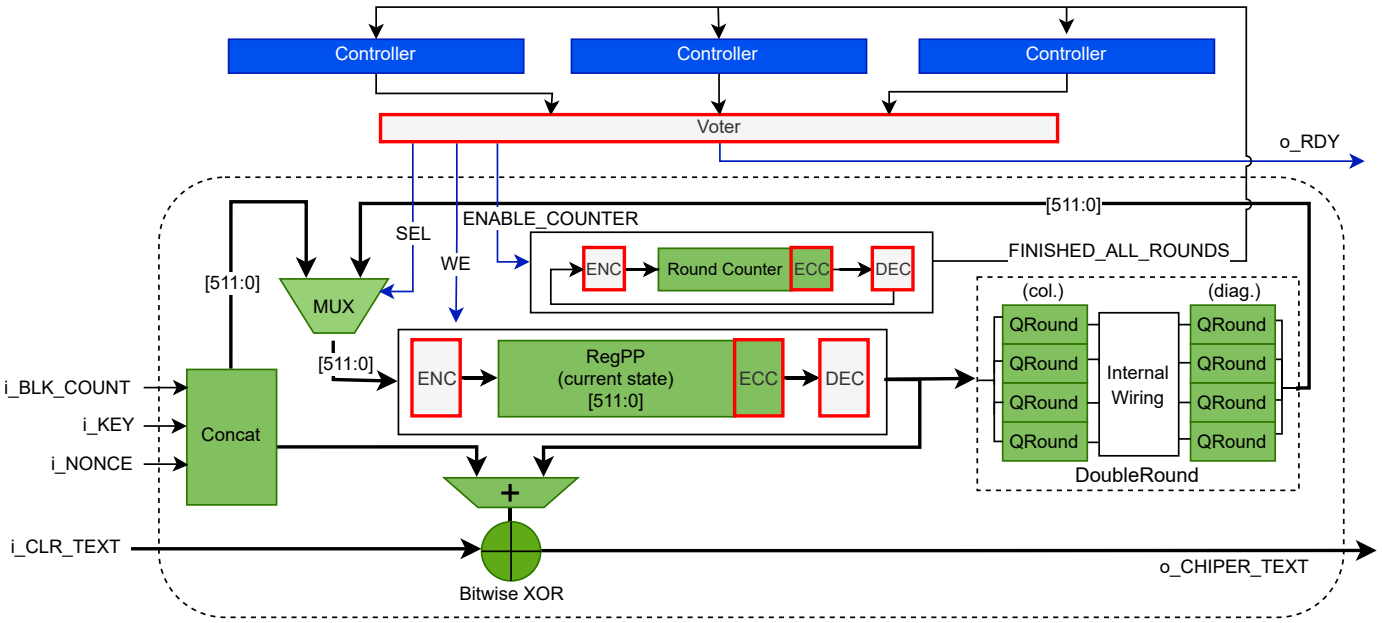

Fig. 1. Block Diagram of the hardened ChaCha20 Accelerator

B. Fault Tolerance Implementation

We applied Hamming ECC to harden the memory elements of the datapath (`RegPP` and `Round Counter`). From Fig. 1, the `ENC` and `DEC` components represent the encoding and decoding logic from Hamming and generate the ECC that is stored with its respective value. The controller was hardened by using TMR with a bitwise majority voter system, sufficient to protect against SEUs affecting one of its triplicated modules, mitigating most of the errors in memory elements for radiation environments, as seen in [11].

C. Fault Injection

We performed a simulation-based fault injection to evaluate the reliability of the accelerator. The solution presented in [12] was used to perform SEU injections affecting a single bit into the registers and was customized to operate on the designed accelerator. This fault injection technique uses built-in commands of the ModelSim simulator.

The fault injection strategy for each iteration consists of initially simulating without fault injections to obtain a golden run. The next stage consists of listing all the registers in the circuit and randomly choosing one bit to perform the fault injection. Next, a time is randomly calculated to perform the fault injection within the simulation execution time. In each simulation, a single fault is injected by inverting a bit within the value of the target signal. Whether the output of any external port differed from the golden run at the end of the simulation, it is assumed that the fault resulted in an error in the ciphertext. We performed 1000 simulations, each configured to encrypt 1MB of data.

III. Results

We used Xilinx Vivado 2020.1 to synthesize the hardware described in VHDL and the Mentor ModelSim 20.1 for design verification and reliability analysis. We used the Xilinx Zedboard development kit (Zynq-7000) for prototyping and the OpenSSL [13] as a software implementation reference.

A. ChaCha20 Accelerator

Table I summarizes the usage of Look-Up Tables (LUTs) and Flip-Flops (FFs), with the maximum operating frequency and estimated power dissipation for each implementation.

TABLE I
ChaCha20 accelerator synthesis results.

Implementation	LUTs	FFs	Fmax (MHz)	Power (mW)
ChaCha20 STD	1,562	1,044	81.80	167
ChaCha20 OPT	2,676	518	52.64	202
ChaCha20 FT	4,086	532	35.09	212

The optimized (OPT) solution requires about 1.63× more LUTs and 2× less FFs than the standard (STD) solution, and presents a lower maximum frequency because the circuit modifications increased the critical path. The hardened (FT) version requires around 52% more LUTs and 3% more FFs than the optimized version, resulting in the hardened version having a 33% reduction in the maximum frequency.

Considering the number of cycles required to encrypt 512 bits of data, we estimated the throughput and energy consumed to process 1MB in each version, as shown in Table II.

TABLE II
ChaCha20 accelerator performance.

Implementation	Cycles	Throughput (MB/s)	Energy* (mJ)
ChaCha20 STD	103	51.25	3.26
ChaCha20 OPT	12	281.25	0.72
ChaCha20 FT	12	187.50	1.13

* Estimated energy consumed to process 1MB.

Even with a lower frequency than the standard solution, the optimized solution achieves a 5.4× higher throughput while consuming 4.6× less energy. The throughput of the hardened version is lower than the optimized version due to the reduction in frequency, but it is still 3.6× higher than the standard version, consuming 2.9× less energy.

B. ChaCha20 SoC

The accelerator was integrated into an SoC to compare the processing time with the software implementation on an ARM processor (Table III). We used the AXI4-Stream interface with a Direct Memory Access (DMA) controller for integration.

TABLE III
ChaCha20 SoC synthesis results.

Implementation	LUTs	FFs	Fmax (MHz)	Power (W)
ChaCha20 SoC STD	3,109	4,391	72.84	1.696
ChaCha20 SoC OPT	3,914	3,871	44.60	1.702
ChaCha20 SoC FT	5,416	3,895	31.78	1.711

The introduction of DMA caused an increase of 2× in LUTs and 4.2× in FFs for the standard solution, and by 1.5× in LUTs and 7.4× in FFs for the optimized and hardened versions. Including DMA with the AXI4-Stream interface slightly reduced the maximum frequency compared to the standalone accelerator.

We compared the SoC versions of the accelerator to the software solution running at 667 MHz on the ARM processor. The system was initially configured to perform the cryptography only by the ARM processor. Then, the ARM processor configures the DMA to send the 512-bit blocks to the accelerator, which controls the cryptography process. Table IV presents the execution time for encrypting 1MB of data.

TABLE IV
Execution time to cipher 1MByte.

Implementation	Execution time (ms)	Acceleration
Software (ARM)	140.13	-
ChaCha20 STD	25.28	5.54
ChaCha20 OPT	5.87	23.87
ChaCha20 FT	8.02	17.47

The standard and optimized solutions show an acceleration of approximately 5.5× and 23× compared to the application entirely in software, and the hardened version shows an acceleration of 17×. The acceleration differs due to the lower maximum frequency reported in the hardened version compared to the optimized one.

C. Reliability analysis

The reliability evaluation consisted of ModelSim simulations executed on a computer with an Intel Core i7-12700 processor and 16 GB of RAM, running a Linux operating system. We conducted 1000 simulations for the optimized and hardened solutions, consuming about 62 hours to validate both configurations. Table V shows the simulation results for the fault injection campaign.

TABLE V
Reliability results.

Implementation	Runs	Errors	Sim. Time
ChaCha20 OPT	1000	918	9h17min
ChaCha20 FT	1000	0	52h35min

TABLE VI
Cost and performance results of the accelerator in comparison with related works.

Work	Implementation	Device	LUTs	FFs	Fmax (MHz)	Power (mW)	Throughput (MB/s)
[8]	-	Virtex-7 XC7VX485T	2,369	2,152	362.50	-	270.00
[9]	Verilog	Cyclone V	1,440	1,094	50.00	-	126.25
[10]	-	Virtex-7 XC7VX485T	2,288	1,050	-	-	-
This work STD	VHDL	Zynq-7000	1,562	1,044	81.80	167	51.25
This work OPT	VHDL	Zynq-7000	2,676	522	52.64	202	281.25
This work FT	VHDL	Zynq-7000	4,086	538	35.09	212	187.50

The simulation results from the optimized version showed 918 errors in 1000 performed simulations. We observed that most of the errors were caused by faults injected into the `RegPP` component, which represents the most significant part of the memory elements in the circuit.

In the hardened version, the implementation detected and corrected all the faults. This demonstrates that the applied Hamming ECC and TMR techniques can protect the circuit against SEUs consisting of single-bit flips.

D. Discussion

The comparison with related works is presented in Table VI. Compared to [8], our optimized solution achieves higher throughput, utilizing only 8% more LUTs and consuming 4× fewer FFs, despite operating at a frequency 7× lower.

When compared to [9], our optimized version achieves a throughput 2.2× higher while using half the FFs. Additionally, the authors emphasize that two accelerator units were required to achieve the reported throughput, and [9] employs software for a portion of the cipher, resulting in a lower LUT count.

The work [10] utilizes a comparable number of LUTs and roughly double the FFs compared to our optimized version, but they do not provide any performance metrics. It is important to mention that the related works do not present hardened solutions.

Despite a decrease in throughput compared to the optimized version, our hardened solution achieves a 1.5× higher throughput than [9] even with a lower frequency. In addition, it is important to mention that the related works do not present hardened solutions.

Regarding resource utilization, our hardened implementation uses 4× fewer FFs than [8] and half the FFs compared to [9], [10]. This resource reduction in FFs represents a valuable result regarding reliability facing SEUs, since memory elements are considered one of the most sensitive parts of circuits in the space environment [14]. Moreover, unlike Single Event Transients (SETs), in FFs, the bit-flips remain stored and potentially create problems up to the FF update.

Our optimized implementation achieves the highest throughput compared to related works implemented in FPGA, even when utilizing a mid-range FPGA device. Additionally, the lower resource utilization of FFs in optimized and hardened solutions sets our approach apart from all other related works.

IV. Conclusion

This paper presented the implementation of a standard and an optimized encryption accelerator for the ChaCha20 stream cipher. We also applied reliability techniques such as Hamming ECC and TMR to generate a fault-tolerant accelerator version.

We performed a fault injection campaign to evaluate the reliability of the accelerator. We observed that the techniques applied were sufficient to protect from SEUs affecting a single bit. We also compared the cost and performance achieved with the related works. Our optimized solution presented higher throughput with a good trade-off between resource utilization.

For future work, we plan to perform a reliability analysis of the accelerator through particle accelerator tests and integrate it into a reliable SoC solution for space applications. Finally, implementing the Poly1305 message authentication code (MAC) would increase the applicability of the accelerator.

References

[1] P. Panahi, C. Bayılmış, U. Çavuşoğlu, and S. Kaçar, “Performance evaluation of lightweight encryption algorithms for iot-based applications,” *Arabian Journal for Science and Engineering*, vol. 46, pp. 4015–4037, 2021.
[2] Y. Zhang, S. Zhao, J. He, Y. Zhang, Y. Shen, X. Jiang *et al.*, “A survey of secure communications for satellite internet based on cryptography and physical layer security,” *IET Information Security*, vol. 2023, 2023.
[3] L. G. de Alvarenga, *Criptografia Clássica e Moderna*. Clube de Autores, 2011.
[4] L. E. Kane, J. J. Chen, R. Thomas, V. Liu, and M. Mckague, “Security and performance in iot: A balancing act,” *IEEE access*, vol. 8, pp. 121 969–121 986, 2020.
[5] D. J. Bernstein *et al.*, “Chacha, a variant of salsa20,” in *Workshop record of SASC*, vol. 8, no. 1, 2008, pp. 3–5.
[6] A. L. Yoav Nir, “Chacha20 and poly1305 for ietf protocols,” Internet Requests for Comments, IRTF, RFC 4180, 03 2018. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8439
[7] V. Krasnov, “It takes two to chacha (poly),” *Cloudflare*, 2016. [Online]. Available: https://blog.cloudflare.com/it-takes-two-to-chacha-poly/
[8] J. Pfau, M. Reuter, T. Harbaum, K. Hofmann, and J. Becker, “A hardware perspective on the chacha ciphers: Scalable chacha8/12/20 implementations ranging from 476 slices to bitrates of 175 gbit/s,” in *2019 32nd IEEE International System-on-Chip Conference (SOCC)*. IEEE, 2019, pp. 294–299.
[9] I. Semenov, “An implementation of chacha20 stream cypher in all-programmable socs,” Ph.D. dissertation, The University of Alabama in Huntsville, 2020.
[10] R. Serrano, C. Duran, T.-T. Hoang, M. Sarmiento, A. Tsukamoto, K. Suzaki, and C.-K. Pham, “Chacha20-poly1305 crypto core compatible with transport layer security 1.3,” in *2021 18th International SoC Design Conference (ISOCC)*. IEEE, 2021, pp. 17–18.
[11] D. A. Santos, A. M. P. Mattos, D. R. Melo, and L. Dilillo, “Characterization of a fault-tolerant RISC-V system-on-chip for space environments,” in *2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2023, pp. 1–6.
[12] D. R. Melo, C. A. Zeferino, L. Dilillo, and E. A. Bezerra, “Maximizing the inner resilience of a network-on-chip through router controllers design,” *Sensors*, vol. 19, no. 24, p. 5416, 2019.
[13] The OpenSSL Project Authors, “Openssl,” 2022. [Online]. Available: https://github.com/openssl/openssl/tree/master/crypto/chacha
[14] D. J. Sorin, *Fault Tolerant Computer Architecture*. Morgan and Claypool Publishers, 2009.