A Dependable and Low-Cost CCSDS 123 Hyperspectral Image Compressor

Wesley Grignani, Douglas A Santos, Felipe Viel, Luigi Dilillo, Douglas R Melo

Title: A Dependable and Low-Cost CCSDS 123 Hyperspectral Image Compressor

Author(s): Wesley Grignani, Douglas A. Santos, Felipe Viel, Luigi Dilillo, and Douglas R. Melo.

DOI: https://doi.org/10.1109/LES.2024.3420934

Document version: Pre-print version (Submitted draft)

Please cite the original version:
W. Grignani, D. A. Santos, F. Viel, L. Dilillo and D. R. Melo, "A Dependable and Low-Cost CCSDS 123 Hyperspectral Image Compressor," in IEEE Embedded Systems Letters, doi: 10.1109/LES.2024.3420934.

A Dependable and Low-Cost CCSDS 123 Hyperspectral Image Compressor

Wesley Grignani, Douglas A. Santos, Felipe Viel, Luigi Dilillo, and Douglas R. Melo

Abstract

One of the most critical challenges in applications that use hyperspectral image (HSI) is the demand for compression, which affects restrictions on the storage capacity and processing in space applications. 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. This work aimed to implement a fault-tolerant CCSDS 123 HSI compressor using a Hardware Description Language (HDL) and fault-tolerant techniques like Triple Modular Redundancy (TMR) and Hamming Error Correcting Code (ECC). A fault injection campaign verified the reliability of the techniques. Results show that the implementation accelerated the application by 24× compared to the software solution. The standard solution can process 20.57 MSa/s, and the hardened solution can process 13.81 MSa/s using 2.2× more Look-Up Tables (LUTs) and 1.4× more Flip-Flops (FFs). The low cost observed in the results makes this implementation a suitable solution for application in space systems targeting resource-efficient devices.

Index Terms

Systems-on-Chip, Hardware Accelerator, Hyperspectral Images, CCSDS 123.0-B-2, Fault Tolerance.

I. Introduction

Space applications that collect information about Earth use remote sensing methods. Among these techniques is hyperspectral imaging, which proves valuable for Earth image acquisition, climate analysis, and forest environments surveillance [1]. HSIs (Hyperspectral Images) are essentially three-dimensional arrays of pixels, with each pixel defined within a (x, y, z) coordinate system. In this context, the z-axis signifies distinct layers corresponding to individual images captured across various parts of the electromagnetic spectrum.

The large number of bands involved in computing HSIs presents difficulties in storing, processing, and transmitting imagery to ground stations. To address the high data volume of HSIs and optimize performance in transmission and processing, the Consultative Committee for Spatial Data Systems developed the CCSDS 123 compression algorithm [2], designed for a low complexity implementation in hardware with lossless and near-lossless compression in the latest release (B-2).

Space systems operate in harsh conditions with radiation, extreme temperatures, vacuum, and low gravity. These challenges can cause temporary, permanent, or intermittent faults, impacting computational systems [3]. To enhance the reliability of space systems, various fault tolerance techniques can be employed to protect them from radiation effects [4].

Some hardware accelerators for lossless HSI compression focus on low resource utilization [5] or high performance [6]. Regarding fault-tolerant solutions, the works [7] and [8] focused on a hardened solution using SG (Space-Grade) FPGAs (Field Programmable Gate Array), where [8] implements Error Detection and Correction. The work [9] performed a reliability analysis of the compressor made in [8], protecting the implementation by triplicating the entire compressor.

In this work, we extend our previous project [10], focusing on implementing a low-cost and fault-tolerant CCSDS 123 hardware accelerator for lossless HSI compression. The accelerator was designed with a combination of spatial and information redundancies to improve the accelerator reliability, including the Hamming ECC (Error Correcting Code) to protect memory elements and the TMR (Triple Modular Redundancy) to protect the control units.

II. CCSDS 123 Accelerator Design

In our previous project [10], we implemented the compressor in HDL (Hardware Description Language) and HLS (High-Level Synthesis) using the AXI4-Lite communication bus. This work uses the HDL implementation and hardens the architecture with fault-tolerance mechanisms. We also improved the SoC (System-on-Chip) communication architecture using an AXI4-Stream bus. The compression design is based on the lossless part of the CCSDS 123.0-B-2 standard with a predictor and an encoder component detailed in the specification [2]. Fig. 1 presents an overview of the accelerator.

Manuscript received April 19, 2005; revised August 26, 2015. This work was supported in part by the Foundation for Support of Research and Innovation, Santa Catarina (FAPESC - grants 2021TR001907 and 2023TR000880), the Brazilian National Council for Scientific and Technological Development (CNPq - processes 138179/2021-2, 140368/2021-3 and 350794/2023-5), and the Region d'Occitanie and the École Doctorale I2S from the University of Montpellier (contract 20007368/ALDOCT-000932).

The authors are with the Laboratory of Embedded and Distributed Systems, University of Vale do Itajaí, Itajaí, Brazil, and the Institut d'Électronique et des Systèmes, University of Montpellier, CNRS, Montpellier, France.
(e-mail: wesley.grignani@edu.univali.br, drm@univali.br)

The prediction calculation is separated by sub-steps, such as local sum $\sigma_{z,y,x}$, local difference $d_{z,y,x}$, weights $W_z(t)$ and local difference $U_z(t)$ vectors, central prediction $\hat{d}_z(t)$, high resolution predicted sample $\breve{s}_z(t)$, double resolution predicted sample $\tilde{s}_z(t)$ and mapped quantizer index $\delta_z(t)$.

The encoder design follows the sample-adaptive model. The output form of the variable-length words can change according to the compressor calculations. To encode a mapped residual $\delta_z(t)$, the model calculates the index $k_z(t)$ based on the values of the predictor and the accumulator $A_z(t)$ and counter $C_z(t)$ memories to generate a Golomb-Power-of-2 word $\Re_k(\delta_z(t))$. During this process, the memories are updated according to the z-band of the image encoded on the sample.

III. Hardware Implementation

A. Design and Implementation Choices

We made some design choices to minimize the use of logic resources. This includes the local sum $\sigma_{z,y,x}$ in column-oriented mode with BIP (Band Interleaved by Pixel) processing orders, allowing the vector size of local differences $U_z(t)$ to be based only on the number of previous bands defined P_z, reducing resources compared to other processing orders. The size of weights vectors $W_z(t)$ is based on the number of Nz bands in the image. The size of the sample buffering is based on the dimensions Nx and Nz of the image according to the BIP order and column-oriented local sum. We used the reduced prediction mode to avoid the additional calculations and storage of the directional local differences of the full mode [11]. These design choices focus on low use of resources by choosing low-complexity design steps.

B. Hardware Architecture

The components were designed to work sequentially, and some steps have been parallelized to reduce the number of cycles needed to process a sample. In addition, as the encoder block only needs the output from the predictor and some image information to encode a sample, it can work in a pipeline with the predictor block.

The local sum and local difference execute simultaneously. The predicted sample and weights update run together after double resolution. The new local difference in the vector can be stored at any step after central prediction and was scheduled to run with the weight update. We created local differences using registers to allow reading all values in a single cycle rather than sequentially from RAM. The weights vectors were created using block RAMs, each using the default initialization method considering the P_z previous bands. As one sample of each band is processed at a time, the z index drives the selection of the weight vector that will be used.

C. Fault Tolerance

We applied Hamming ECC to protect each register between the stages in the predictor and the encoder. Hamming ECC was implemented to the weights and local difference vectors in the predictor and the accumulator and counter vectors in the encoder block.

Both the predictor and encoder controllers were hardened by using TMR with a simple bit-wise majority voter system, sufficient to protect against SEUs (Single Event Upsets) affecting a single bit, mitigating most of the errors in memory elements for radiation environments, as seen in [12]. In Section IV, we present a reliability analysis of the implementation.

IV. Results

Initially, the implementation was validated with the simulation results from a default pattern test image provided by CCSDS. The final compression result was compared with [13] to validate the accelerator function. Images from different sensors like Landsat [14], AVIRIS [15], Hyperion [16], and HICO [17] were also used to verify the accelerator performance.

The Xilinx Vivado 2020.1 development tool was used for synthesis and performance results, considering the standard and the hardened versions of the compressor.

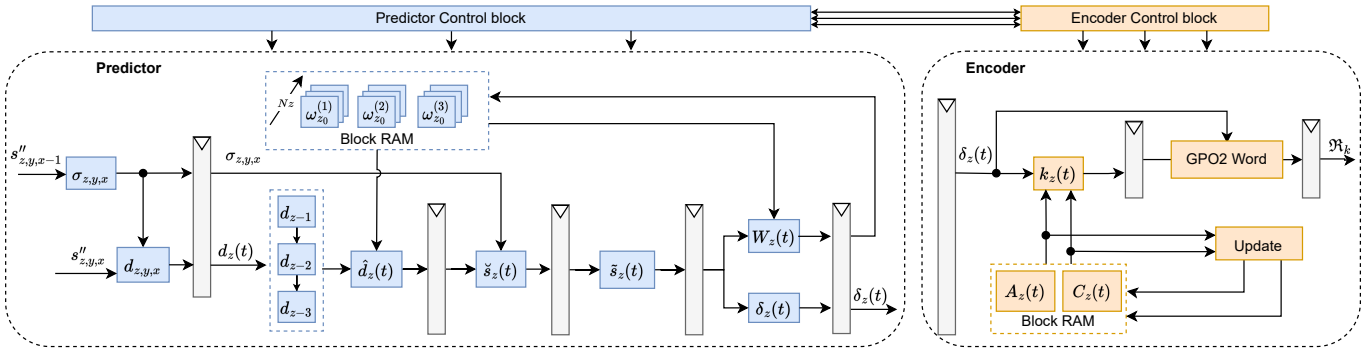

Fig. 1. Compressor design overview.

A. Synthesis Results

Table I shows the CCSDS 123 parameters used for synthesis and verification analysis. Many of these parameters impact the image compression rate, except for those parameters that also impact the use of resources (Ω, P_z, R). A detailed analysis of the impact of these parameters is given in [11].

TABLE I
CCSDS PARAMETERS USED FOR SYNTHESIS RESULTS.

Parameter	Description	Value
D	Dynamic Sample size in bits	16
Ω	Weight Component Resolution (Weight size is Ω + 3)	8
v_{min}	Weight Update Scaling Exponent Initial Parameter	5
v_{max}	Weight Update Scaling Exponent Final Parameter	9
t_{inc}	Weight Update Scaling Exponent Change Interval	2^6
P_z	Number of Previous Bands Used in Prediction	3
R	Register size	32
U_{max}	Unary length limit	16
$\gamma 0$	Initial count exponent	1
γ^*	Re-scaling counter size	4
K	Accumulator Initialization Constant	5

TABLE II
RESOURCE UTILIZATION AND PERFORMANCE FOR STANDARD AND HARDENED COMPRESSOR.

Sensor	Nx	Ny	Nz	Config.	LUTs	FFs	DSPs	BRAMs	Fmax (MHz)	Power (mW)	Throughput[1] (MSa/s)
Landsat	1024	1024	6	standard	1167	507	4	6	98.69	119	19.74
				hardened	2567 (+120%)	687 (+36%)	4	8 (+33%)	70.20	144	14.04 (-29%)
AVIRIS	680	512	224	standard	1314	511	5	82	100.49	146	20.09
				hardened	2918 (+122%)	722 (+41%)	5	107 (+31%)	66.54	176	13.31 (-33%)
Hyperion	256	3242	242	standard	1294	519	4	34	102.89	129	20.57
				hardened	2845 (+120%)	710 (+37%)	4	43 (+27%)	69.05	147	13.81 (-32%)
HICO	500	2000	87	standard	1274	501	5	34	100.98	124	20.19
				hardened	2819 (+121%)	714 (+42%)	5	44 (+29%)	73.32	156	14.66 (-27%)

[1] A Sample represents one pixel of the image.

Table II presents the compressor resource utilization and performance for different sets of images. The configurations are referred to as 'hardened', representing the compressor with TMR and Hamming hardening, and 'standard', without hardening. The values of LUTs and FFs change because the sizes Nx, Ny, and Nz of the image change the register width to control the image index internally. The biggest difference in resource use between the images concerns using BRAMs.

The use of BRAMs increases according to the Nx and Nz size of the image according to the necessary buffering for BIP processing order. Between the standard implementations, the Landsat image configuration required the lowest resources, especially because of the lowest number of bands. On the other hand, the AVIRIS image presented the highest use of BRAMs and LUTs compared to other standard implementations; in particular, AVIRIS showed 13$\times$ the use of BRAMs and 1.13$\times$ the use of LUTs when compared with Landsat.

The compressor takes 5 cycles to process one sample. The highest throughput achieved in the standard implementations was 20.57 MSa/s for the Hyperion configuration. As the compressor architecture does not change with different image configurations, the number of cycles to process a sample remains the same. Therefore, the difference in throughput is due to the maximum frequency according to the synthesis tool for each image, where the *Performance_ExploreWithRemap* strategy was set.

In the hardened version, resource utilization increases in all image configurations by an average of 120% in LUTs, 39% in FFs, and 30% in BRAMs. This results from applying TMR in the controllers and Hamming to all the compression stages. In addition, the throughput decreased by an average of 30% since the maximum frequency of the circuit also decreased. This frequency drop is mainly due to additional Hamming encoding and decoding logic that increased the critical path in the compression stages.

B. SoC Results

The compressor was tested on the Zynq-7020 SoC (XC7Z020-CLG484) with AXI DMA to measure the acceleration of the compression application. The system was initially configured to perform image compression only by the ARM Cortex-A9 processor in a software solution. Then, the ARM processor generates the image header and configures the DMA to send the samples to the compressor.

The accelerators were configured based on image size and maximum frequency from AVIRIS. The HSI compression application was accelerated by an average of 24$\times$ in the standard versions and by 16$\times$ in the hardened ones compared to the ARM processor running at 667 MHz. The solution accelerates the application by moving the routines from software to dedicated hardware, and the standard version accelerates even more because of its frequency and throughput.

C. Reliability analysis

We used a simulation method to conduct the fault injection campaign using an AVIRIS image cropped into 4 sizes. We performed 1000 simulations for each image configuration, where a single fault was injected in a random memory element at a random time within the simulation time. Results were compared against a golden run conducted without fault injection. Table III presents the simulation results.

TABLE III
SIMULATION RESULTS FOR DIFFERENT IMAGE SIZES.

Nx	Ny	Nz	Standard	Sim. time	Hardened	Sim. time
20	20	20	541	10min	0	31min
20	20	224	727	1h40min	0	4h52min
128	128	20	512	6h07min	0	19h29min
128	128	112	654	34h56min	0	103h44min

Results show that the standard configuration presents several errors that vary according to the image dimensions. This can be explained by the weight size and buffer memories growing accordingly. They represent most memory elements in the system, thus becoming the most probable target of fault injection. Considering the error propagation, we observed that any bit-flip in the weight value directly affects the result because its value is recalculated constantly, propagating each error to the next samples. On the other hand, the buffer has its values overwritten by new samples at every iteration. This reduces the error propagation but can still cause errors if the fault is injected before the sample is used.

The results also show that the hardened implementation could detect and correct faults. This demonstrates that the Hamming ECC and TMR techniques can protect the circuit against SEUs consisting of single-bit flips. In addition, the applied techniques resulted in a lower overhead when compared to triplicating the entire core.

TABLE IV
SYNTHESIS AND PERFORMANCE RESULTS IN COMPARISON WITH RELATED WORKS.

Work	Hardened	FPGA	LUTs	FFs	DSPs	BRAMs	Fmax (MHz)	Power (mW)	Throughput[1] (MSa/s)	t_{exe}[2] (ms)	Energy[2] (mJ)
[5]	No	Zynq-7000	2244	630	3	-	142.00	106	20.40	49.02	5.22
[6]	No	Zynq-7020	3012	2528	6	84	147.00	295	147.00	6.80	2.00
[7]	SG FPGA	Virtex-5 FX130T	9462	9990	6	83	213.00	-	213.00	4.69	-
[8]	Partially	Zynq-7035	4619	2765	8	74	151.10	-	151.10	6.62	-
[9]	DMR	Zynq-7020	12878	7327	12	21	63.00	69	63.00	15.87	1.09
[9]	TMR	Zynq-7020	19589	10986	18	31	62.80	144	62.80	15.92	2.29
This work	No	Zynq-7020	1294	519	4	34	102.89	129	20.57	48.61	6.24
This work	TMR + Ham	Zynq-7020	2845	710	4	43	69.05	147	13.81	72.41	11.48

[1] A Sample represents one pixel of the image. [2] Execution time and maximum estimated energy consumed to process 1MSa.

D. Comparison with Related Works

Table IV presents the results of the standard and hardened versions with related works. The standard implementation has the lowest resource utilization of any other related work. Compared to standard works, our implementation uses 1.7× fewer LUTs and 1.2× fewer FFs than [5], which only implements the compression prediction step. On the other hand, our implementation uses 2.3× fewer LUTs and 4.8× fewer FFs than [6], but they have a throughput 7× higher than this work.

Compared to hardened works, our solution uses 3.3× fewer LUTs and 14× fewer FFs than [7] and 1.6× fewer LUTs and 3.8× fewer FFs than [8]. These works have a higher throughput due to the pipelined and parallelized architecture. Moreover, the work [7] does not present hardened circuitry while [8] implements EDAC (Error Detection And Correction) in some internal memories to protect from SEUs.

The work [9] uses the DMR (Dual Modular Redundancy) and the TMR technique to implement the hardened solution. Our work presents a resource utilization of about 4.5× less LUTs and 10× less FFs in the DMR version and uses 6× less LUTs and 15× less FFs in the TMR version. While our work applies fault tolerance internally to compression components, the techniques applied in [9] duplicate and triplicate the entire core and significantly increase resource utilization.

Due to the high throughput, the related works have a lower execution time than this work, except for [5], which presents a similar performance compared with the standard solution.

This work focuses on fault-tolerant and low-cost implementation, and it does not have higher throughput than related works. However, despite a decrease in throughput, it is worth mentioning that the resource reduction, especially 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 [4].

V. CONCLUSION

This paper presents an implementation of a CCSDS 123 lossless HSI compressor described in HDL. The implementation was hardened to be fault-tolerant and low-cost.

The compressor does not implement optimization techniques, so it does not present the best performance results. On the other hand, the implementation presents the lowest cost compared to the related works, which is ideal for resource-constrained devices for space systems.

We compared the achieved performance with standard implementations and works that implemented protection. Unlike the related works, we applied fault tolerance techniques to all internal memories and control units in the hardened version. The results show good resilience against SEUs because of lower resource utilization.

For future work, we plan to extend the fault injection campaign with different images and perform an experimental analysis in particle accelerators. We also intend to improve the execution time performance of the compressor by implementing a pipelined architecture and integrating dependable multi-core systems.

REFERENCES

[1] F. Viel, R. C. Maciel, L. O. Seman, C. A. Zeferino, E. A. Bezerra, and V. R. Q. Leithardt, "Hyperspectral image classification: An analysis employing CNN, LSTM, Transformer, and Attention Mechanism," *IEEE Access*, vol. 11, pp. 24 835–24 850, 2023.

[2] CCSDS, "Low-complexity lossless and near-lossless multispectral and hyperspectral image compression," Available at: https://public.ccsds.org/Pubs/123x0b2c3.pdf. Accessed: February 06, 2024, p. 102, 2021, cor.3.

[3] M. Yang, G. Hua, Y. Feng, and J. Gong, *Fault-tolerance techniques for spacecraft control computers*. John Wiley & Sons, 2017.

[4] D. J. Sorin, *Fault Tolerant Computer Architecture*. Morgan and Claypool Publishers, 2009.

[5] L. M. Pereira, D. A. Santos, C. A. Zeferino, and D. R. Melo, "A low-cost hardware accelerator for CCSDS 123 predictor in FPGA," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[6] J. Fjeldtvedt, M. Orlandić, and T. A. Johansen, "An efficient real-time FPGA implementation of the CCSDS-123 compression standard for hyperspectral images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 10, pp. 3841–3852, 2018.

[7] A. Tsigkanos, N. Kranitis, G. Theodorou, and A. Paschalis, "A 3.3 Gbps CCSDS 123.0-B-1 multispectral & hyperspectral image compression hardware accelerator on a space-grade SRAM FPGA," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 1, pp. 90–103, 2018.

[8] Y. Barrios, A. J. Sánchez, L. Santos, and R. Sarmiento, "SHyLoC 2.0: A versatile hardware solution for on-board data and hyperspectral image compression on future space missions," *Ieee Access*, vol. 8, pp. 54 269–54 287, 2020.

[9] L. A. Aranda, A. Sánchez, F. Garcia-Herrero, Y. Barrios, R. Sarmiento, and J. A. Maestro, "Reliability analysis of the SHyLoC CCSDS123 IP core for lossless hyperspectral image compression using COTS FPGAs," *Electronics*, vol. 9, no. 10, p. 1681, 2020.

[10] W. Grignani, D. A. dos Santos, L. Dilillo, F. Viel, and D. R. de Melo, "A low-cost hardware accelerator for CCSDS 123 lossless hyperspectral image compression," in *DFT 2023-36th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, 2023.

[11] I. Blanes, A. Kiely, M. Hernández-Cabronero, and J. Serra-Sagristà, "Performance impact of parameter tuning on the CCSDS-123.0-B-2 low-complexity lossless and near-lossless multispectral and hyperspectral image compression standard," *Remote Sensing*, vol. 11, no. 11, p. 1390, 2019.

[12] 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.

[13] GICi, "GICI emporda CCSDS 123.0-B-1," Available at: http://gici.uab.cat/GiciApps/EmpordaManual.pdf. Accessed: February 9, 2024, p. 15, 2011.

[14] NASA, "Landsat," Available at: https://landsat.gsfc.nasa.gov/. Accessed: February 9, 2024.

[15] ——, "Airborne Visible InfraRed Imaging Spectrometer," Available at: https://aviris.jpl.nasa.gov/. Accessed: February 9, 2024.

[16] USGS, "Hyperion - USGS EO-1," Available at: https://eo1.usgs.gov/sensors/hyperion. Accessed: February 9, 2024.

[17] M. D. Lewis, R. Gould, R. Arnone, P. Lyon, P. Martinolich, R. Vaughan, A. Lawson, T. Scardino, W. Hou, W. Snyder *et al.*, "The Hyperspectral Imager for the Coastal Ocean (HICO): Sensor and data processing overview," in *OCEANS 2009*. IEEE, 2009, pp. 1–9.