On-board Compressing of Hyperspectral Images using CCSDS 123

Douglas A. Santos
University of Vale do Itajaí
Santa Catarina, Brazil
douglasas@edu.univali.br

Cesar A. Zeferino
University of Vale do Itajaí
Santa Catarina, Brazil
zeferino@univali.br

Eduardo A. Bezerra
Federal University of Santa Catarina
Santa Catarina, Brazil
eduardo.bezerra@ufsc.br

Luigi Dilillo
Federal University of Santa Catarina
Santa Catarina, Brazil
dilillo@lirmm.fr

Douglas R. Melo
Federal University of Santa Catarina
Santa Catarina, Brazil
drm@ieee.org

ABSTRACT

A satellite performing hyperspectral image processing requires high storage capacity and larger communication bandwidth. Compression algorithms, like the CCSDS 123, have been proposed to mitigate these requirements. Considering the constraints associated to satellites, single-purpose processors have been developed to run these algorithms in Systems-on-Chip (SoC). In this work, we evaluate alternatives to integrate a CCSDS 123 compressor with an embedded processor based on RISC-V and ARM-based architectures.

KEYWORDS

Hyperspectral Images, Compression, System-on-Chip, RISC-V

1 INTRODUCTION

Hyperspectral images are large data arrays that describe the electromagnetic spectrum. They can be applied in several areas, such as medicine, biogeochemistry, biophysics, and industrial monitoring [1]. These images usually include hundreds of bands, producing a huge amount of data.

The compression of hyperspectral images can be a helpful feature in remote sensing systems [2]. For this purpose, the Consultative Committee for Space Data Systems (CCSDS) [3] has proposed a recommendation for lossless compression of hyperspectral images, called CCSDS 123. This recommendation benefits from the characteristics of the hyperspectral images to increase the compression rate. The recommendation defines a throughput of 20 MSa/s (Samples per Second) as the minimum rate to achieve real-time. Therefore, some works have developed hardware accelerators to reduce power consumption and to increase communication throughput.

The authors in [4] developed an accelerator for the prediction stage of the CCSDS algorithm, which is the most costly one. The implementation focused on the low use of logical elements in an FPGA (Field Programmable Gate Array) device.

Since the space environment is susceptible to faults due to ionizing particles, which can cause failures in the system, fault tolerance techniques are required to protect these systems. According to [5], Single Event Transient (SET) and Single Event Upset (SEU) are effects with a high occurrence probability in integrated circuits applied in this type of environment. A SET fault happens when ionizing particles hit the device, and it experiences a transient pulse interference in its wires and logic. The SEU fault takes effect when the logical value of a memory element suffers an inversion as a result of the impinging particle.

In this context, the paper presents a comparative study between two types of integrations of a hardware-accelerated predictor system, which on one side implements a hard-core processor (ARM-based), and the other a soft-core (RISC-V-based) processor. This study aims at evaluating the architectural parameters and resources that are employed in the two implementations to identify their criticalities. For example, essential parameters include critical paths, which interests the occurrence of delay faults and SETs, and the amount of used sequential logic that is particularly prone to SEUs. This analysis supports further integration of the co-processor in a reliable version of the RISC-V, which is under development by our research team.

2 BACKGROUND

This section presents the background regarding this work, by introducing concepts on hyperspectral images, the CCSDS 123 standard, and the computer system architectures used for the evaluation.

2.1 Hyperspectral Images

Hyperspectral images have ample spectral information with several wavelengths that allows the identification and distinguishing of spectrally similar materials [6]. Fig. 1 presents a visual representation of a hyperspectral image.

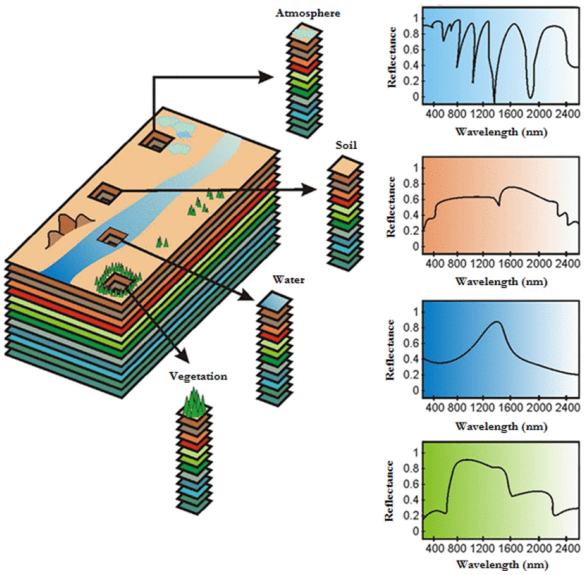

Figure 1: Hyperspectral Image Representation [7]

Hyperspectral images are extensively used in remote sensing, due to its potential for more accurate and detailed information extraction when compared to other types of remotely sensed data [6]. According to [8], hyperspectral imagery combines two sensing modalities: imaging, related to the spatial distribution, and spectrometry, which measures the power variation with the wavelength or frequency of light.

An example of the hyperspectral image is scene 0 from the uncalibrated image YellowStone of the [9] database. Fig. 2 presents a visual representation in the RGB spectrum of this scene.

Figure 2: RGB Yellowstone Scene 0 [9]

2.2 CCSDS 123

CCSDS 123 is a lossless compression algorithm recommendation defined by the CCSDS committee. This algorithm was explicitly created for hyperspectral images, taking advantage of its characteristics [10]. It consists of two stages: prediction and encoding, presented in Fig. 3. The input of the algorithm is a three-dimensional image, and the resulting compressed image is an encoded bitstream, which can reconstruct the input image.

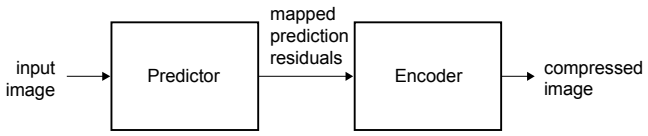

Figure 3: CCSDS overview [10]

The prediction stage is an adaptive linear prediction method based on the values of the neighbor samples from the current and previous bands. The prediction has six steps, which are: local sum, local difference, inner product, predictor, mapped, and weight update.

In this work, we are using the column-oriented local sum, which results is four times the sample of the previous line when $y > 0$ and four times the sample of the previous column when $y = 0$ and $x > 0$. The local difference adopted is the central-local difference, which value is four times the current sample minus the equivalent local sum.

The inner product steps are the calculation of the inner product between the local differences of the three previous bands and the three weights of the current band. After this step, the scaled predicted sample is calculated based on the inner product, local sum, and the current sample. The mapped prediction residual is the result of the prediction stage and relates itself to the difference between the scaled predicted sample and the sample value.

The second stage is responsible for creating the output of the compressor. Initially, it outputs the meta-data of the compressed image, which contains relevant information for its decompression. After, the algorithm processes each mapped result from the predictor. The process consists of an entropy encoder that losslessly encodes the predictor output and sends it to the compressor output.

2.3 System Architectures

We have been elaborating on two different approaches — the first implementation hard-core ARM-based. The ARM (Advanced RISC Machine) architecture is one of the most licensed processor cores in the world. The ARMv8 architecture uses micro-architecture techniques to provide small and efficient implementations [11].

Because of the reduced instruction set, the ARM architecture requires fewer transistor than CISC (Complex Instruction Set Computer) processors, enabling smaller die size of integrated circuits. This architecture has three profiles: application profile, real-time profile, and microcontroller. Many consumer electronics adopt ARM processors, such as smartphones, tablets, multimedia players, and wearables.

The second approach used in this work uses a soft-core RISC-V system. RISC-V is an open instruction set architecture (ISA) supported by the RISC-V foundation [12]. It has as the main principle to be adequate to any computational device, being adaptable to a wide range of applications, from high-performance devices to simple microcontrollers.

The RISC-V architecture has been designed to simplify the hardware implementation, using a very regular instruction encoding, and a direct memory model with no complex instructions for memory access. One of the benefits of the RISC-V is that minimal cores are much smaller than similar ones as, for instance, ARM and x86, although the difference is not meaningful in higher performance cores [13]. Fig. 4 presents an overview of a simple single-cycle RISC-V architecture, which is very similar to the MIPS.

PULPino [14] is a single-core System-on-Chip (SoC) based on RISC-V architecture that can use RI5CY or zero-riscy cores. Fig. 5 shows a block diagram of the SoC with an AMBA (Advanced Microcontroller Bus Architecture) AXI (Advanced eXtensible Interface) as its main bus and a bridge to APB (Advanced Peripheral Bus) for peripherals. Both buses feature 32-bit wide data channels.

The RI5CY [15], used by PULPino, is a 32-bit RISC-V core with a 4-stage pipeline, implemented with the focus on ISA and micro-architecture optimization, specifically targeting parallel applications.

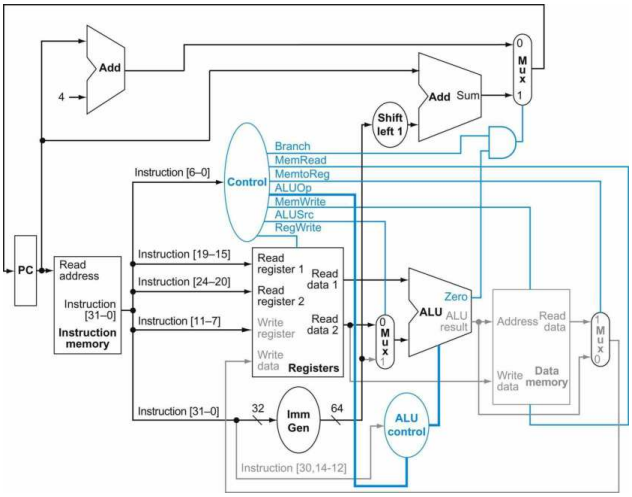

Figure 4: RISC-V overview [12]

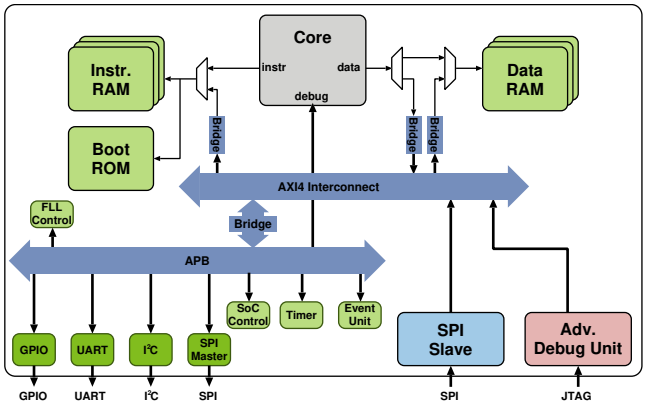

Figure 5: PULPino overview [14].

3 PROPOSED ARCHITECTURES

Among the stages of the compression process, the prediction is the most costly one as a result of the intensive use of arithmetic operations. A prior work [4] implemented an accelerator for this stage on FPGA, focusing on low usage of logical resources and achieving the minimum requirement for real-time processing.

The authors applied the column-oriented local sum approach and the reduced prediction mode, which achieves better performance with raw images in comparison with the use of calibrated images. To allow sharing the weights among various instances of the core, they implemented the weights memory outside the core.

The samples are processed in the BIP (Band Interleaved by Pixel) format, reducing the amount of required memory. The compressor implementation (Fig. 6) predicts one sample every seven cycles with a maximum operating frequency of 142.85 MHz.

The implementation of the core has enabled real-time processing with a low cost in hardware. According to [4], their processor achieved a maximum of 20.4 MSa/s, requiring 561 FPGA slices. Thus, allowing the designer to obtain a good trade-off between efficiency and logical resources.

In this work, we are performing the integration of the predictor core to the SoC architectures through an AMBA bus. The Fig 7 presents a visual representation of the wrapper. Since the co-processor is running at a higher frequency than the main processor, we had to implement the block Predictor enable, which has the purpose of keeping the enable flag available during only one predictor clock cycle. Using this scheme, we ensure that the co-processor processes the sample only once.

The samples data required by the processor come from the main processor, which is stored by the Input Register. After the predictor finishes its tasks, it writes the mapped result to the Output Register and activates a ready flag. The general-purpose processor is responsible for the retrieval and forwarding of the samples from the data memory to the co-processor.

The image compression flow that runs in the main processor is presented in Algorithm 1. The first *for* loop initializes the counter and accumulator, required by the encoding stage. The second *for* loop runs three steps to get the sample for the compression: (*i*) set sample, which gets the current sample data retrieved from the data memory; (*ii*) set neighbour, which receives the neighbour specified by the CCSDS recommendation; and (*iii*) set start flag, which sends the start flag to the co-processor. After these steps, it waits until the co-processor finishes the execution of the prediction, encoding the mapped result.

Algorithm 1 CCSDS 123 Compression.

```
1:  procedure MAIN
2:      for each band do
3:          initializecounter[band]
4:          initializeaccumulator[band]
5:      end for
6:      for each sample do
7:          set sample
8:          set neighbour
9:          set start flag
10:         unset start flag
11:         while not ready do
12:             wait
13:         end while
14:         encode_mapped(sample, counter, accumulator)
15:     end for
16: end procedure
```

Our work has integrated the predictor co-processor with two different architectures: a hard-core ARM and a soft-core RISC-V. In Sections 3.1 and 3.2, we describe these implementations.

ARM integration

We used the hard-core ARM available in the FPGA device. This integration was made using only Xilinx proprietary IPs. The predictor is connected using the previously described wrapper to connect to an AMBA AXI bus, using the AXI interconnect Xilinx IP. The ARM processor and the bus are running at 660 MHz and 100 MHz, respectively.

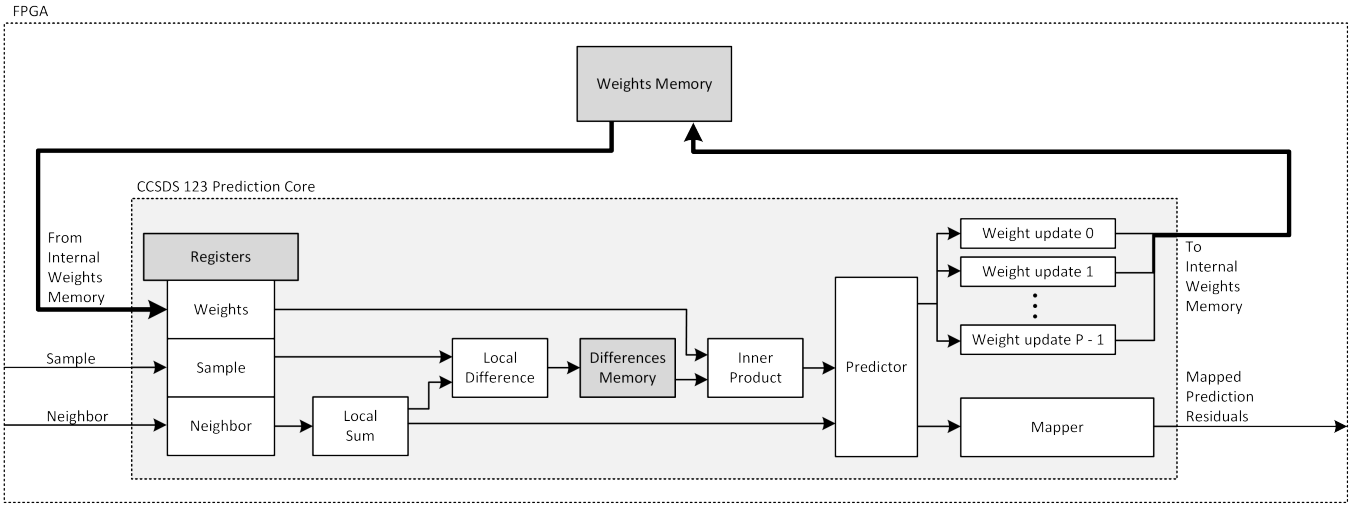

Figure 6: Compressor block diagram [4].

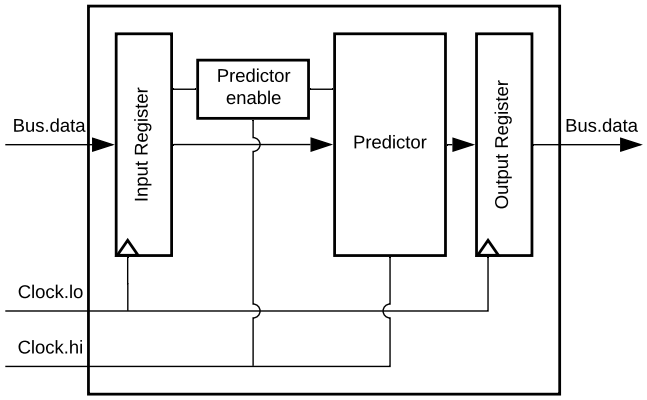

Figure 7: Predictor Co-processor Wrapper.

RISC-V integration

The integration with RISC-V was made using a soft-core version. We chose PULPino [14] as it is widely adopted and has an implementation of the AMBA bus. The predictor co-processor is connected to the peripheral bus (AMBA APB).

The PULPino core runs with a 5 MHz clock by default, as well as the AMBA buses. This architecture is implemented entirely on the FPGA logical resources, using discrete parts only to essential connections, like clock generators and GPIO (General-Purpose Input/Output).

4 VERIFICATION AND RESULTS

Our implementations were validated using the Zynq-7000 SoC from the Zedboard development kit. The image used is the Yellowstone uncalibrated Scene 0, from the AVIRIS database [9], presented in 2.1. The compressed image with both integrations was the same compared to the one from the Empordá software [19].

Table 1 presents a comparison between the new implementation described in this paper and related work. When comparing the previous implementation [4] to our proposal, we noticed that the maximum operating frequency is slightly different, this is due to the Vivado synthesis tool estimation, as the metrics authors from previous work obtained the frequency through FPGA prototyping.

The logical resource usage differs in each integration. While the ARM one uses 2,387 LUTs, 1,091 FFs, and 3 DSPs, the RISC-V integration uses 18,890 LUTs, 12,392 FFs, and 9 DSPs. The higher quantity of logical resources used by the second one is the consequence of using a soft-core processor instead of a hard-core one. The Programmable Logic (PL) was used instead of the Processing System (PS).

Using the Vivado software power analysis, we obtained a dynamic power of 0.607 W and 0.245 W when employing the ARM and the RISC-V implementations, respectively. RISC-V has lower power figures than ARM as the entire SoC with RISC-V runs at a much lower clock frequency.

Regarding the performance in each integration, the tool estimated that when using the PULPino SoC, the maximum frequency (F_{max}) of the predictor co-processor is 55.94 MHz. At the same time, with the ARM, we obtained 61.56 MHz. The difference is due to the type of implementation in which the logical elements and connections are placed in a more optimized way when there are more available resources (ARM), thus directly influencing critical paths of the circuit. A longer critical path is not only a performance issue, but it makes the system more prone to delay faults and SETs.

As there is no optimization in the communication between the processor and the co-processor, these systems cannot reach the real-time requirement, presenting a lower throughput when compared to related works.

Table 1: Comparison with related works

Work	FPGA	Slices	LUTs	FFs	DSPs	Throughput (MSa/s)	F_{max} (MHz)	Mode	Predictor	Encoder
[2]	Virtex-5 FX130	834	*n.a.*	*n.a.*	*n.a.*	55.4	55.4	Full	HW	*n.i.*
[16]	Virtex-5 VFX130	842	2,342	1,535	1	11.3	43.4	Full	HW	HW
[17]	Virtex-7 XC7	24,238*	96,955	*n.a.*	25	219.4	*n.a.*	Full	HW	HW
[18]	Zynq-7000	3,008*	12,033	10,696	28	750	157	Full	HW	HW
[4]	Zynq-7000	561*	2,244	630	3	20.4	142.85	Reduced	HW	*n.i.*
ARM-based	Zynq-7000	597*	2,387	1,091	3	1.43	61.56	Reduced	HW	SW
RISC-V-based	Zynq-7000	4,722*	18,890	12,392	9	0.088	55.94	Reduced	HW	SW

Notes: *n.i.*: not implemented; *n.a.*: information not available; *estimated values (1 slice = 4 LUTs)

5 CONCLUSIONS

In this work, we integrated and evaluated a predictor co-processor by using two different implementations, with ARM-based SoC (hard-core) and RISC-V (soft-core) architectures. The maximum operating frequency when integrated with RISC-V is lower because the predictor shares the programmable logic part with the RISC-V, causing an increase of the critical paths, which affects the susceptibility to SET faults and delay faults. Also, the soft-core RISC-V implementation requires more sequential logic, such as registers, then affecting the sensitivity to SEU faults. Unlike the RISC-V, the ARM core used in this work is not fully customizable, which restricts the possibility of applying fault tolerance techniques and also to improve the reliability in the processor-level.

As future work, we intend to integrate the predictor with a fault-tolerant version of the RISC-V processor and evaluate the reliability of the system with neutrons radiation test. That processor is currently under development in our research group, focusing on the application in reliable systems, such as CubeSats.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) – Processes 315287/2018-7 and 436982/2018-8. The authors also thank for the support of CSU – University Space Center of Montpellier, France.

REFERENCES

[1] Muhammad Mateen, Junhao Wen, Nasrullah, and Muhammad Azeem Akbar. The role of hyperspectral imaging: A literature review. *International Journal of Advanced Computer Science and Applications*, 9(8):51–62, 2018.

[2] G. Lopez, E. Napoli, and A. G. M. Strollo. Fpga implementation of the ccsds-123.0-b-1 lossless hyperspectral image compression algorithm prediction stage. In *2015 IEEE 6th Latin American Symposium on Circuits Systems (LASCAS)*, pages 1–4. IEEE, Feb 2015. doi: 10.1109/LASCAS.2015.7250438.

[3] Consulative Committee for Space Data Systems (CCSDS). *Lossless Multispectral and Hyperspectral Image Compression.* Green Book. CCSDS, Washington, DC, 2015. URL https://public.ccsds.org/Pubs/120x2g1.pdf.

[4] L. M. V. 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)*, pages 1–5. IEEE, May 2019. doi: 10.1109/ISCAS.2019.8702428.

[5] Mengfei Yang, Gengxin Hua, Yanjun Feng, and Jian Gong. *Fault-tolerance techniques for spacecraft control computers.* Wiley Publishing, 1 Fusionopolis Walk, #07-01 Solaris South Tower, Singapore 138628, 1st edition, 2017. ISBN 111910727X, 9781119107279.

[6] Peg Shippert et al. Why use hyperspectral imagery? *Photogrammetric engineering and remote sensing*, 70(4):377–396, 2004.

[7] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza. Fpga implementation of the n-findr algorithm for remotely sensed hyperspectral image analysis. *IEEE Transactions on Geoscience and Remote Sensing*, 50(2):374–388, Feb 2012. doi: 10.1109/TGRS.2011.2171693.

[8] Michael Theodore Eismann. *Hyperspectral remote sensing.* SPIE Bellingham, Bellingham, Washington 98227-0010 USA, 01 2012. doi: 10.1117/3.899758.

[9] JPL. Hyperspectral image compression, 2006. URL https://coding.jpl.nasa.gov/hyperspectral/.

[10] CCSDS. Lossless multispectral and hyperspectral image compression, May 2012. URL https://public.ccsds.org/Pubs/123x0b1ec1s.pdf.

[11] ARM. Architecture reference manual: Armv8, for armv8-a architecture profile, 2019.

[12] David Patterson and Andrew Waterman. *The RISC-V reader: An open architecture atlas.* Strawberry Canyon, 2017. ISBN 099924910X, 9780999249109.

[13] David Kanter. Risc-v offers simple, modular isa. *The Linley Group MICROPROCESSOR Report (March 2016)*, pages 1–5, 2016.

[14] A. Traber and M. Gautschi. Pulpino: Datasheet, jun 2017. URL https://pulp-platform.org/docs/pulpino_datasheet.pdf.

[15] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini. Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, Oct 2017. doi: 10.1109/TVLSI.2017.2654506.

[16] L. Santos, L. Berrojo, J. Moreno, J. F. López, and R. Sarmiento. Multispectral and hyperspectral lossless compressor for space applications (hyloc): A low-complexity fpga implementation of the ccsds 123 standard. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(2):757–770, Feb 2016. ISSN 1939-1404. doi: 10.1109/JSTARS.2015.2497163.

[17] Daniel Báscones, Carlos Gonzalez, and Daniel Mozos. Parallel implementation of the ccsds 1.2.3 standard for hyperspectral lossless compression. *Remote Sensing*, 9:973, 09 2017. doi: 10.3390/rs9100973.

[18] Milica Orlandić, Johan Fjeldtvedt, and Tor Arne Johansen. A parallel fpga implementation of the ccsds-123 compression algorithm. *Remote Sensing*, 11(6):673, 2019.

[19] J. E. Sanchez, E. Auge, J. Santalo, I. Blanes, J. Serra-Sagrista, and A. Kiely. Review and implementation of the emerging ccsds recommended standard for multispectral and hyperspectral lossless image coding. In *2011 First International Conference on Data Compression, Communications and Processing*, pages 222–228. IEEE, June 2011. doi: 10.1109/CCP.2011.17.