electronics

MDPI

Article

Lossy Hyperspectral Image Compression on a Reconfigurable and Fault-Tolerant FPGA-Based Adaptive Computing Platform †

Yubal Barrios [1,*], Alfonso Rodríguez [2], Antonio Sánchez [1], Arturo Pérez [2], Sebastián López [1], Andrés Otero [2], Eduardo de la Torre [2] and Roberto Sarmiento [1]

1 Institute for Applied Microelectronics (IUMA), University of Las Palmas de Gran Canaria, 35001 Las Palmas de Gran Canaria, Spain; ajsanchez@iuma.ulpgc.es (A.S.); seblopez@iuma.ulpgc.es (S.L.); roberto@iuma.ulpgc.es (R.S.)

2 Centre of Industrial Electronics, Universidad Politécnica de Madrid, 28006 Madrid, Spain; alfonso.rodriguezm@upm.es (A.R.); arturo.perez@upm.es (A.P.); joseandres.otero@upm.es (A.O.); eduardo.delatorre@upm.es (E.d.l.T.)

* Correspondence: ybarrios@iuma.ulpgc.es

† This paper is an extended version of our paper published in "Hyperspectral Image Lossy Compression on a Reconfigurable and Fault-Tolerant Architecture Implemented over a COTS FPGA-Based System-on-Chip" presented at the 6th International Workshop on On-Board Payload Data Compression (OBPDC 2018).

Received: 25 August 2020; Accepted: 22 September 2020; Published: 26 September 2020

Abstract: This paper describes a novel hardware implementation of a lossy multispectral and hyperspectral image compressor for on-board operation in space missions. The compression algorithm is a lossy extension of the Consultative Committee for Space Data Systems (CCSDS) 123.0-B-1 lossless standard that includes a bit-rate control stage, which in turn manages the losses the compressor may introduce to achieve higher compression ratios without compromising the recovered image quality. The algorithm has been implemented using High-Level Synthesis (HLS) techniques to increase design productivity by raising the abstraction level. The proposed lossy compression solution is deployed onto ARTICo^3, a dynamically reconfigurable multi-accelerator architecture, obtaining a run-time adaptive solution that enables user-selectable performance (i.e., load more hardware accelerators to transparently increase throughput), power consumption, and fault tolerance (i.e., group hardware accelerators to transparently enable hardware redundancy). The whole compression solution is tested on a Xilinx Zynq UltraScale+ Field-Programmable Gate Array (FPGA)-based MPSoC using different input images, from multispectral to ultraspectral. For images acquired by the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS), the proposed implementation renders an execution time of approximately 36 s when 8 accelerators are compressing concurrently at 100 MHz, which in turn uses around 20% of the LUTs and 17% of the dedicated memory blocks available in the target device. In this scenario, a speedup of 15.6× is obtained in comparison with a pure software version of the algorithm running in an ARM Cortex-A53 processor.

Keywords: hyperspectral imaging; lossy data compression; hardware acceleration; on-board processing; CCSDS; fault-tolerance

1. Introduction

The use of hyperspectral sensors on-board satellites is taking relevance for environmental studies. In the last years, Earth Observation (EO) space missions are incorporating this kind of sensors with identification and detection purposes. Besides, spectroscopy is proposed to be integrated into deep-space missions to analyse cosmic bodies, such as the surface of Mars [1]. Their main disadvantage

is that acquired images demand too many memory resources to be stored because they include the complete spectral information in a range of wavelengths for every single pixel. This tremendous amount of data prevents from sending them directly to ground stations due to downlink bandwidth limitations [2]. Therefore, on-board image compression techniques become a must to reduce image size before sending them to the on-Earth facilities where they are processed.

In the image compression state-of-the-art, two categories are distinguished: lossless and lossy compression. Lossless compression preserves all the data of the acquired image, fully recovering it after decompression. Consequently, the achieved compression ratios are limited (typically 4:1, or even less) and insufficient for large-distance space missions with high-resolution hyperspectral sensors on-board. On the other hand, lossy compression reaches higher compression ratios introducing losses in the data; i.e., the original image cannot be recovered, obtaining an approximation that depends on the applied error threshold [3]. Although getting a specific compression ratio is the primary goal, the computational complexity of the compression algorithm must also be considered, taking into account the limited computational resources available on-board.

Currently, on-board compression algorithms are implemented on Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs). FPGAs are increasing their presence as part of space payloads because of their reconfiguration capabilities, high performance, low-power consumption and cost reduction compared to ASICs. In addition, FPGAs allow to execute simultaneously certain operations if no data dependencies are presented, supposing an advantage compared to the sequential behaviour of embedded microprocessors. FPGAs also have the ability to change all or some parts of the functionality dynamically to adapt it to new requirements that can appear during the mission lifetime or even if corruption takes place, because of radiation effects. Although there are Rad-Hard By Design (RHBD) FPGAs available in the market, specially designed for working in critical environments under radiation, Commercial Off-The-Shelf (COTS) SRAM-based FPGAs are gaining increasing interest due to their higher performance in comparison with RHBD devices, which belong to former technological generations.

These FPGAs can be combined with microprocessors in the same die, forming heterogeneous Systems-on-Chips (SoCs), which increase system capabilities in terms of computational performance without compromising its power consumption. The space industry is integrating these SoCs in SmallSats for Low-Earth Orbit (LEO) missions where the radiation effects are reduced, to demonstrate the viability of this technology for short-time experiments [4]. As it is well-known, radiation can cause errors in the FPGAs configuration memory, which may derive in a progressive or even complete malfunctioning of the device. Nevertheless, some techniques, such as scrubbing and hardware redundancy at different levels, are currently available to prevent or mitigate radiation effects and to preserve data integrity and system functionality [5].

FPGA design has been addressed in the last decades at Register Transfer Level (RTL), but with the increasing design complexity, it is necessary to use other approaches that allow the reduction of both the development time and the re-design costs. In this way, High-Level Synthesis (HLS) methodologies can be used, starting from an algorithmic model in a high-level language that is automatically transformed by HLS tools into its equivalent RTL description [6]. The generated RTL model should match the design constraints defined at higher abstraction levels to achieve the goals in terms of timing, resources and power consumption. If these objectives are not achieved, it is possible to come back to a previous stage and modify specific parts of the algorithm with total flexibility and without additional manufacturing costs. Moreover, this design flow considers the possibility of working at RTL, if particular targets in terms of clock-cycling accuracy are required.

In this paper, we propose the implementation of a lossy extension of the CCSDS 123.0-B-1 lossless compression standard [7] for multispectral and hyperspectral images. This extension consists in adding a quantizer and a bit-rate feedback loop, to control the losses for achieving the targeted compression ratios without deteriorating in excess the quality of the decompressed image. This solution has been developed using HLS techniques, and it is implemented over a reconfigurable, scalable

and fault-tolerant architecture named ARTICo^3 [8], which provides run-time adaptive computing performance, energy efficiency, and robustness against radiation when used as a payload processing system in short-time missions with COTS devices. As a result, a configurable number of lossy compressor cores can be placed in the FPGA fabric and then set to work not only with different input data to maximise performance and energy efficiency, but also with the same data to maximise fault mitigation by using hardware redundancy techniques. A brief introduction about this compression solution ans some preliminary results can be found in [9].

The rest of this paper is organised as follows. A review of the state-of-the-art in hyperspectral image compression is provided in Section 2. Section 3 provides an overview of the CCSDS 123.0-B-1 lossless standard and the proposed extension to work in the near-lossless and lossy modes. The ARTICo^3 framework is presented in Section 4, while the mapping of the compression application onto this architecture is detailed in Section 5, together with an impact study about the influence of the image partitioning in the quality of the reconstructed image. Then, experimental results are shown in Section 6, including timing, area and compression results. Finally, Section 7 summarizes the conclusions and the future work.

2. Related Work

Hyperspectral image compression is a hot topic in the space industry and, consequently, there is a significant amount of research works available in the specialised literature.

Regarding compression algorithms, the preference of the scientific community is the use of lossless solutions, to preserve as much essential information as possible from the acquired scene. The implementation of this kind of algorithms, generally based on a predictive stage, must take into account important constraints, such as low-complexity and low-power consumption, in order to fit well with the limited computational resources available on-board satellites. In this category, the CCSDS 123.0-B-1 standard must be highlighted [7], as it provides a low-complexity solution for on-board compression of multispectral and hyperspectral images. Different implementations of this algorithm on FPGAs can be found in the state-of-the-art, in both RHBD [10–15] and COTS devices [16,17], and dealing with different strategies to optimise either the throughput or the resources consumption. All the implementations mentioned above refer only to one compression instance; however, works described in [10,15,17] and further extended in [18–20], respectively, follow a parallel approach, where multiple compression instances work simultaneously, improving the results in terms of throughput. Finally, it is also possible to find some implementations of this algorithm on GPUs such as [21,22], although their high power consumption and lack of tolerance to ionising radiation make GPUs unfit for space applications.

However, if higher compression ratios need to be achieved in order to reduce the on-board storage demands, lossy compression solutions should be adopted. Generally, lossy algorithms are transform-based, such as the Discrete Wavelet Transform (DWT) [23] or the Karhunen–Loève Transform (KLT) [24], the latter being the one that provides the best results in terms of rate-distortion [25]. Despite the suitable results presented, the KLT approach has some disadvantages, such as a high computational and memory demands, its higher implementation costs and its lack in terms of scalability, preventing its use for specific applications with strong constraints, like on-board compression. The Pairwise Orthogonal Transform (POT) derives from the KLT but reducing the complexity of the operations and still obtaining at the same time better results than the Wavelet-based approaches [26]. The CCSDS has also defined the CCSDS 122.1-B-1 spectral preprocessing transform for 3D image compression [27], based on the Integer Wavelet Transform (IWT), the same used by the JPEG2000 standard, and an encoding stage where multiple instances of a 2D encoder apply a DWT to a band of the downshifted transformed image. In addition, there are other lossy algorithms in the literature [28–32], which try to adapt the complex transform-based solutions to be able to compress hyperspectral images, taking into account the hardware limitations on-board satellites.

An intermediate point between lossless and lossy solutions is the use of near-lossless algorithms, whose complexity is similar to lossless algorithms but at the same time are able to achieve compression ratios in the same order of magnitude than lossy solutions. Recently, the CCSDS has extended the CCSDS 123.0-B-1 lossless algorithm for working in the near-lossless range, introducing a quantizer and a local decompressor for estimating the reconstructed samples in the prediction loop. This new standard, named CCSDS 123.0-B-2 [33], it is based on the Fast Lossless Extended (FLEX) algorithm [34] and defines a low-complexity compression architecture that provides backward compatibility with the lossless counterpart. Still, at the same time, it is also capable of performing lossy compression, providing results in terms of rate-distortion similar to other lossy algorithms. As this standard has been recently published, the implementation presented in [35] is the only one available in the state-of-the-art.

With respect to hardware implementations of near-lossless and lossy compression algorithms for hyperspectral images, some works are found targeting both FPGAs [36–40] and also GPUs [41,42]. However, none of these implementations provides a way to smartly adapt the compression performance under certain anomalous situations, aiming at enhancing the system with extra features, such as robustness against faults defining modular redundancy or reprogramming a specific part of the FPGA in case of complete corruption due to radiation effects. In addition, the work presented in this paper takes as starting point a standard solution, adapting its structure to extend its compression capabilities.

For these reasons, the proposed work represents a new solution for short-time and Low-Orbit missions, providing not only good results in terms of rate-distortion but also including some dynamic techniques to adapt the architecture to the extreme conditions existent in the space environment. This latter feature supposes an added value in comparison with other works available in the specialised literature, at the expense of a small penalty in terms of timing capabilities. In addition, taking into account the results provided in Section 6, the complexity of the presented solution fits well with the hardware resources available on SmallSats, which makes the solution proposed in this paper a convenient option for this kind of satellites.

3. Compression Algorithm Description

The CCSDS 123.0-B-1 lossless compression standard [7] is intended for compressing multispectral and hyperspectral images on-board satellites, achieving a trade-off between compression efficiency and computational demands. The output data is a variable-length encoded bitstream from which the original image can be fully recovered. It comprises two main stages: prediction and entropy coding of the prediction residuals (i.e., the differences between the current sample and the predicted one).

Some of the authors have previously developed SHyLoC [14], a set of two technology-independent Intellectual Property (IP) cores described in VHDL. One of these IP cores implements the functionality described in the CCSDS 123.0-B-1 lossless standard and currently it is part of the ESA IP Cores Library [43]. The high-level model developed as a proof-of-concept for the IP mentioned above is used as the starting point in this work, extending its functionality to work in near-lossless and lossy ranges.

3.1. Predictor

The prediction stage decorrelates redundant information in the acquired image, using a spatial and spectral neighbourhood of previously processed samples to predict the value of the current one, as shown in Figure 1. This decorrelation stage helps to reduce the compressed data size, dealing with differences among pixels instead of the real pixel values. Therefore, if $s_{z,y,x}$ is a sample located in the spatial coordinates (y, x) and band z, the predicted sample $\hat{s}_{z,y,x}$ is calculated using previously pre-processed neighbouring samples in the current and in the P previous bands. The number of previous bands P used for prediction is a tunable parameter between 0 and 15, although it has been demonstrated in [44] that there is not a significant improvement from $P = 3$ onward.

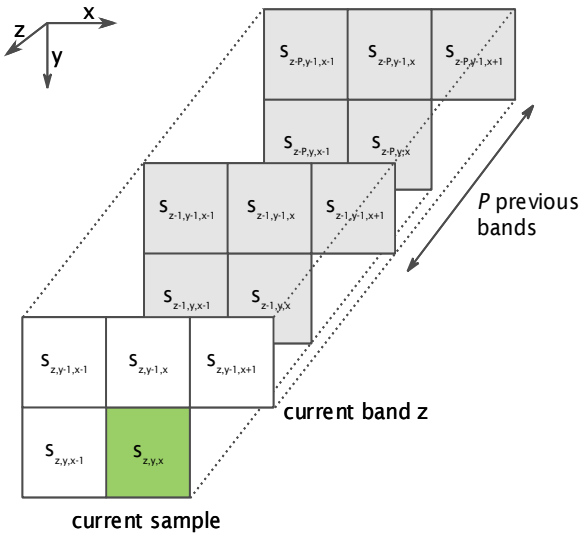

Figure 1. Sample neighbourhood considered for prediction.

The predictor first computes a local sum $\sigma_{z,y,x}$ of neighbouring sample values in the current band as well as for each one of the P previous bands. This computation can be done using either a neighbour- or a column-oriented approach. If the neighbour-oriented mode is selected, all the previously processed adjacent samples are used (see Equation (1)), while in the column-oriented mode just the sample right above is used, as it is reflected in Equation (2).

$$\sigma_{z,y,x} = \begin{cases} s_{z,y,x-1} + s_{z,y-1,x-1} + s_{z,y-1,x} + s_{z,y-1,x+1}, & y > 0, 0 < x < N_x - 1 \\ 4s_{z,y,x-1}, & y = 0, x > 0 \\ 2(s_{z,y-1,x} + s_{z,y-1,x+1}), & y > 0, x = 0 \\ s_{z,y,x-1} + s_{z,y-1,x-1} + 2s_{z,y-1,x}, & y > 0, x = N_x - 1 \end{cases} \tag{1}$$

$$\sigma_{z,y,x} = \begin{cases} 4s_{z,y-1,x}, & y > 0 \\ 4s_{z,y,x-1}, & y = 0, x > 0 \end{cases} \tag{2}$$

These local sums are then used to compute the local differences $d_{z,y,x}$, subtracting the value of the samples in the neighbourhood of the current one from the calculated local sum. The central difference is computed as $d_{z,y,x} = 4s_{z,y,x} - \sigma_{z,y,x}$, while the directional differences are computed according to Equations (3)–(5), if the full prediction mode is selected. Under the reduced mode, only the central difference is considered to shape the differences vector $U_{z,y,x}$.

Then, the predicted sample $\hat{s}_{z,y,x}$ is calculated using the local sum $\sigma_{z,y,x}$ in the current spectral band and a weighted sum of the elements in $U_{z,y,x}$. A weight vector is maintained separately for each band, and its components are updated considering the prediction residual $\Delta_{z,y,x}$ and the local differences $d_{z,y,x}$. Finally, the prediction residual $\Delta_{z,y,x}$ is mapped into an unsigned integer $\delta_{z,y,x}$, the mapped prediction residual, which is subsequently encoded by the selected entropy coder.

$$d^N_{z,y,x} = \begin{cases} 4s_{z,y-1,x} - \sigma_{z,y,x}, & y > 0 \\ 0, & x > 0, y = 0 \end{cases} \tag{3}$$

$$d^W_{z,y,x} = \begin{cases} 4s_{z,y,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & x > 0, y = 0 \end{cases} \tag{4}$$

$$d^{NW}_{z,y,x} = \begin{cases} 4s_{z,y-1,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & x > 0, y = 0 \end{cases} \quad (5)$$

3.2. Entropy Coder

Regarding the encoding stage, the CCSDS 123.0-B-1 standard allows selecting between a sample-adaptive entropy encoder and the block-adaptive entropy encoder defined by the CCSDS 121.0-B-2 lossless compression standard [45]. In this work, the sample-adaptive encoder has been selected because it generally achieves a higher compression ratio than the block-adaptive one [46]. These results are even better in the case of calibrated images, where streaking artefacts do not appear.

The sample-adaptive encoder uses a Golomb power-of-two variable-length binary codeword to encode the mapped prediction residuals $\delta_{z,y,x}$ one by one. The code selected for each input residual depends on some image statistics, represented by a counter and an accumulator that are updated with every new sample according to some user-defined parameters, summarized in Table 1.

Table 1. Sample-adaptive encoder parameters.

Parameter	Allowed Values	Description
U_{max}	[8:32]	Unary Length Limit
$\gamma*$	[max(4,γ_0+1):9]	Rescaling Counter Size
γ_0	[1:8]	Initial Count Exponent
k	[0:D-2]	Accumulator Initialization Constant

The encoded image starts with a header that includes the necessary information to decompress the bitstream, followed by the encoded body itself. This body is generated by encoding the mapped prediction residuals $\delta_{z,y,x}$ with the disposition specified by the user and encoded in the header. Three possible arrangements are considered: Band-Sequential (BSQ) order, where the samples in a band are managed before processing the next one, and Band-Interleaved (BI) order. Two possible orders are identified in this latter case: Band-Interleaved-by-Pixel (BIP), where a sample is processed with all its spectral information before handling the next one, and Band-Interleaved-by-Line (BIL), in which a complete line of samples in the spatial domain is processed before starting the next band. These samples arrangements are reflected visually in Figure 2.

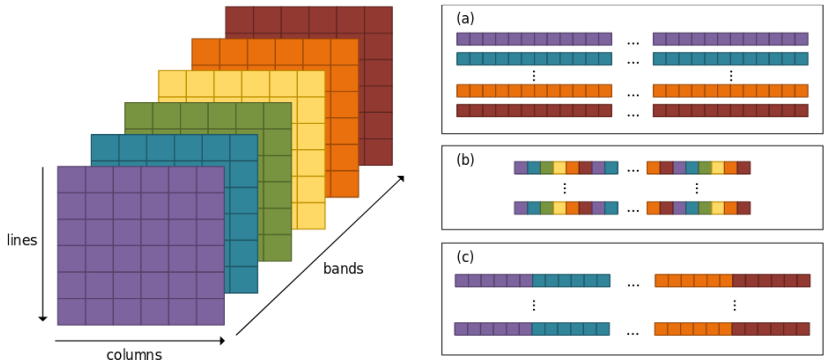

Figure 2. Samples arrangements: (**a**) Band-Sequential (BSQ); (**b**) Band-Interleaved-by-Pixel (BIP); (**c**) Band-Interleaved-by-Line (BIL).

3.3. Lossy Extension

The baseline lossless architecture has been extended to work in a near-lossless to lossy range, adding two extra modules: a quantizer and a bit-rate control. Figure 3 shows an overview of the compression solution after inserting these two modules. The proposed extension is based on the

modifications described in [47,48], which yield higher compression ratios than the lossless solution without incurring penalties in terms of resources and memory usage, but features a novel and hardware-friendly rate control mode.

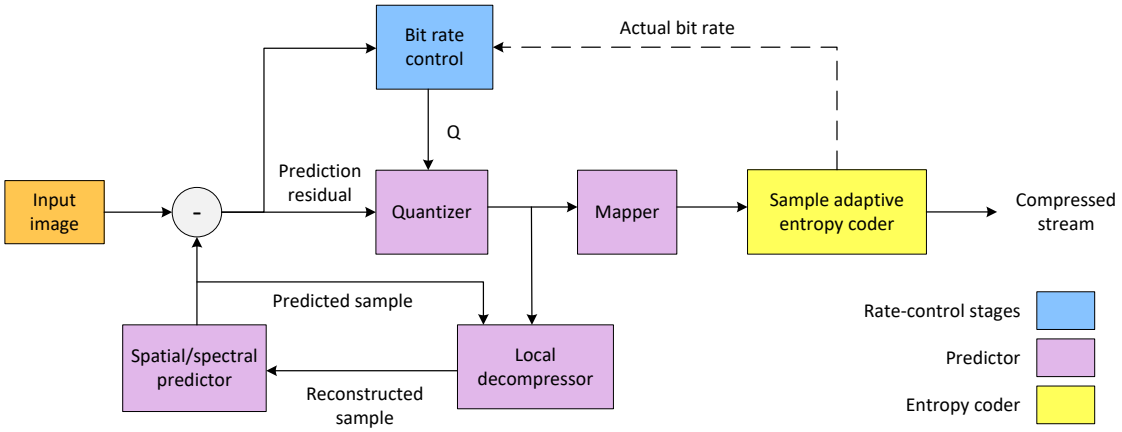

Figure 3. General overview of the lossy solution architecture.

3.3.1. Quantizer

In the proposed architecture, the uniform scalar quantizer from [47] is placed in the predictor architecture prior to the mapping stage. This module provides support for near-lossless compression, taking the prediction residual $\Delta_{z,y,x}$ as input and generating a quantized residual $q_{z,y,x}$ according to Equation (6) (where Q is the quantization step value).

$$q_{z,y,x} = sgn(\Delta_{z,y,x}) \cdot \left\lfloor \frac{|\Delta_{z,y,x}| + (\frac{Q+1}{2})}{Q} \right\rfloor \tag{6}$$

A dequantizer is also necessary to correctly reconstruct the samples before updating the weights in the prediction stage.

3.3.2. Bit-Rate Control

Selection of the Quantization Step Size

The compressor works internally in BIL order, performing the compression line by line, and considering as a spectral line a row of pixels in the spatial domain with all their spectral information. Following the approach proposed in [48], a single quantization step is applied to each spectral line. Depending on the compression rate reached for the current spectral line, the rate control determines the step to be used in the next line to achieve the compression ratio specified by the user.

The calculation of the suitable quantization step Q_z for the next line is done through the statistical characterization of the quantized residuals $q_{z,y,x}$ in each band of the current line, considering that the differences between two adjacent lines are small. Thus, the whole image can be processed in a single pass, enabling a pipeline between the prediction and encoding stages.

In our proposal, the median is used as statistical estimator instead of the mean, because the latter is not robust against outliers, as it is justified in [48]. Regarding the implementation complexity, it must be noted that one estimator per line and band is required. Therefore, in every line, N_z estimators of N_x samples each must be computed. If the mean was to be used as the estimator, a division by N_x would be required, which depends on the input image dimensions.

To compute these medians, different categories are assigned to the image pixels according to their position, as shown in Figure 4. Each category is handled in a particular way along the rate-control process:

- Type A: these quantized residuals are propagated to the sample-adaptive encoder and, at the same time, they are stored in a local memory.
- Type B: after $L-1$ type-A pixels, a type-B pixel is reached. At this point, the median of the whole segment ($L-1$ type-A pixels plus the type-B pixel) is calculated and stored in an array of medians.
- Type C: the end of a row in the spatial domain is denoted with a type-C pixel. At this point, both the median of the last (possibly incomplete) type-A segment and the median of medians for the whole spectral band are computed, storing the latter in a global array.
- Type D: the type-D pixel denotes the end of an image for a given line ($x = N_x$ and $z = N_z$); i.e., it is the type-C pixel of band N_z. At this point, all the medians of medians calculated for each band are used to decide the proper quantization step for the next spectral line.

As it is demonstrated in [48], a value of $L = 17$ represents a good trade-off between the medians calculated at type-B and type-C pixels, without increasing in excess the complexity of the calculations. In our implementation, a segment is stored locally using registers, reducing the latency compared to solutions where memory blocks are used. In the latter case, an extra cycle is needed for each access, either for reading or writing.

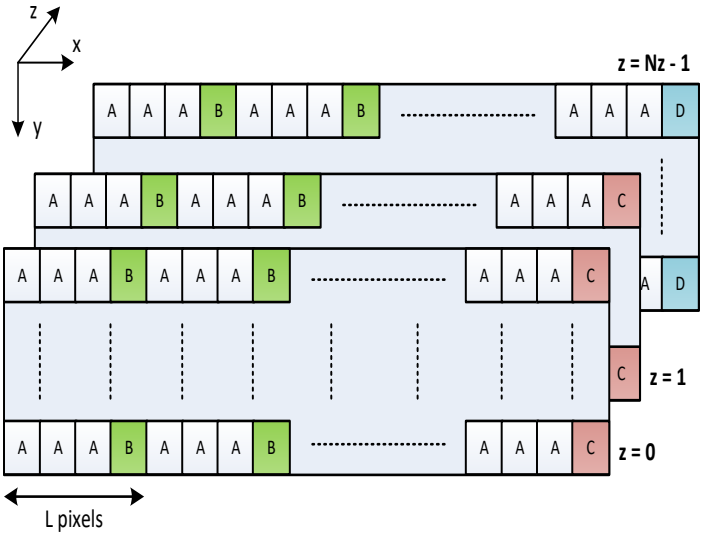

Figure 4. Pixel classification in the lossy compression scheme.

Besides, a sorting function has been included to correctly arrange the residuals of a segment each time a new residual is generated, to reduce latency in the median computation. This is possible because prediction residuals are generated at a rate enough to perform sorting operations in parallel with the processing of the next samples, without incurring in a penalty in terms of throughput. By the time the last prediction residual in a group is generated, it is only necessary to place this value in an already sorted set, which is simpler than a full arithmetic division.

This technique uses a rate LUT to find the suitable quantization step size that provides, for the next spectral line, the closest bit-rate to the target one, based on the statistical characterization of the current line. As a result, a complex computation is substituted by a fast indexing operation, which in turn reduces the complexity of the algorithm. How this lookup table is indexed is also detailed in [48].

The selected quantization steps need to be included in the header of the encoded bitstream to ensure proper image recovery on the decompressor side. Hence, the inclusion of the Q_z values in the header is a non-standard solution that implies an overhead proportional to the number of rows in the image N_y, since a different quantization step value is used per image line. As the maximum quantization step size accepted by our architecture is 511 (this value has enabled very low target bit-rates under experimental evaluation), the overhead of encoding each Q_z is 9 bits, in the worst case.

Feedback Loop

The real bit-rate achieved when compressing a spectral line may differ from the target bit-rate estimated according to the selected quantization step. Two different rate strategies are proposed in [47] to deal with this issue: mode A and mode B. While mode A does not include a feedback loop to use information from previous lines (i.e., achieved bit-rate) to adjust the quantization steps, mode B takes into account these results.

Mode B adjusts the target bit-rate in a line by line basis employing two terms: local correction and global correction. The first term is a function of the difference between target and real bit-rates for the previous line, while the global correction depends on the accumulated bit-rate deviation and a configurable time constant. In addition, both terms make use of an accuracy ratio, computed as the quotient between the real and target bit-rates for a given line, averaged over a configurable number of previous lines.

The bit-rate control proposed in this work is based on the aforementioned mode B, which has been conveniently modified to obtain a hardware-friendly solution with lower complexity and latency. This method, named Simplified-B (SB) mode, is a lightweight version of mode B that avoids complex arithmetic operations not suitable for hardware implementations, such as divisions. In this way, a lower resources utilization is achieved.

In particular, mode B performs divisions to obtain the bit-rate for the current line and also to calculate the local and global corrections. In the SB mode, on the other hand, the global correction is neglected and the accuracy ratio is omitted. Thus, only a simplified local correction is performed taking into account the bit-rate specified by the user and the excess or saving regarding the bit-rate used in the previous spectral line. In this way, the high correlation between spectral lines in a hyperspectral image is exploited. Then, the difference between target and real bit-rates, denoted as the bit-rate error, is computed to adjust the target bit-rate of the next line, together with the selected quantization step value. Therefore, there is a feedback from the sample-adaptive encoder output to the bit-rate control module, adapting the bit-rate by dynamically applying these local corrections.

Finally, it is assumed that the introduction of the feedback loop to control the losses generates a degradation in timing features (especially throughput) since the length of the critical path is extended.

4. The ARTICo^3 Framework

ARTICo^3 is an open source framework for run-time adaptive multi-accelerator system design and management. (Further information in https://des-cei.github.io/tools/artico3) The framework provides three different components: an FPGA-based reconfigurable processing architecture, an automated toolchain to build reconfigurable multi-accelerator systems, and a runtime library to manage device reconfiguration and parallel computation offloading transparently.

The ARTICo^3 architecture [8] is a hardware-based processing architecture for high-performance embedded reconfigurable computing. ARTICo^3 enables run-time adaptive implementations of data-parallel algorithms on SRAM-based FPGAs using Dynamic and Partial Reconfiguration (DPR), a technique that allows online circuit modification and replacement after system deployment (i.e., during execution). Moreover, application adaptivity is user-driven, generating a solution space defined by dynamic trade-offs between computing performance, power consumption and fault tolerance.

As in most DPR-enabled architectures, ARTICo^3 is divided into two different regions: static, which contains the logic resources that are not modified during normal system execution, and dynamic (or reconfigurable), which contains a certain number of slots that can be changed at run time without interfering with the rest of the system. The number, size, and distribution of the reconfigurable slots is specified at design time, and has strong technology dependencies (e.g., FPGA size, logic resources distribution). Despite these restrictions, the architecture is described in target-independent VHDL code, which ensures portability across Xilinx devices.

The communication infrastructure in ARTICo^3 is built around a configurable module that behaves as a bridge between the static region and the reconfigurable multi-accelerator system, enabling efficient data transfers between both domains. The memory-mapped communication with the hardware accelerators varies depending on the purpose of the transactions: application data (i.e., data used for processing) are moved in burst-based transfers, whereas configuration data (i.e., data used for control purposes) are transferred using register-based accesses. The top-level block diagram of the ARTICo^3 architecture is shown in Figure 5.

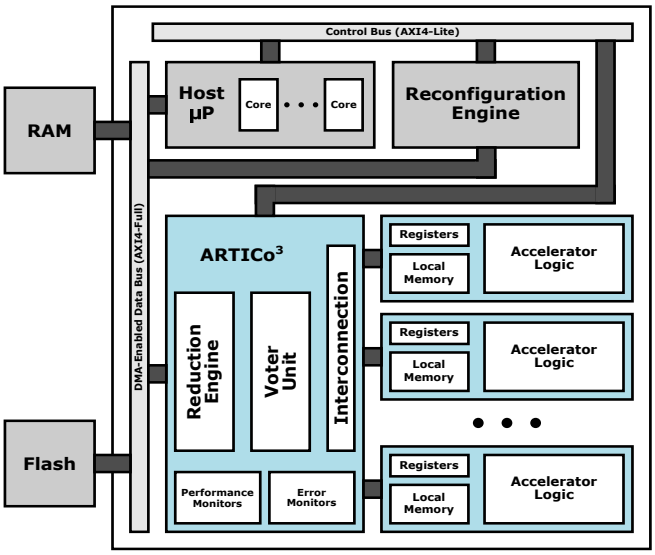

Figure 5. ARTICo^3 architecture overview.

DPR-powered module replication in the reconfigurable partitions (or slots) is combined with an optimized and DMA-capable datapath, whose internal structure can be dynamically modified to support different processing profiles. Hence, performance-oriented or fault-tolerant processing can be selected on-demand. Assuming that multiple copies of a given hardware accelerator have been loaded in the FPGA fabric using DPR, the former would use different input data for each replica to exploit SIMD-like execution. In contrast, the latter would deliver the same input data to each copy and vote the obtained results to mask faults, following redundancy strategies such as Dual or Triple Modular Redundancy (DMR or TMR, respectively) to get a more robust system against radiation in space environments. It is worth noting that these processing profiles enable scalable execution performance (i.e., several accelerators executing in SIMD-like fashion are faster than a single one, provided that there are no bandwidth limitations) and adaptive fault tolerance levels (i.e., three accelerators executing in TMR are more dependable than a single one with no redundancy, while keeping the same performance) during normal system execution. Besides, an embedded monitoring infrastructure allows users to analyze system performance and error rates per slot.

Traditionally, both the design and management of dynamically reconfigurable hardware systems have been complex tasks. Two additional components complement the processing architecture to reduce development effort and make the ARTICo^3 architecture accessible to embedded system engineers with little or no previous experience on hardware design: a toolchain to automate system generation from source code to application binaries, and a runtime library to hide parallelism deployment and DPR from programmers.

ARTICo^3-based processing follows a processor-coprocessor approach, where an application runs on a host processor and data-parallel computations are offloaded to the kernels, implemented as hardware accelerators with a configurable number of registers and memory banks. The ARTICo^3 toolchain takes an already partitioned hardware/software application as input and generates both the executable to be run on the host processor and the configuration files to program the FPGA. While the

host code needs to be specified using C/C++ descriptions, hardware accelerators can be designed using low-level RTL models (for highly optimized and timing-accurate designs) or high-level C/C++ code, since High-Level Synthesis (HLS) is also supported within the toolchain.

Two different types of parallelism can be exploited using ARTICo^3: on the one hand, its execution model provides transparent data-level parallelism by using several replicas of the same hardware accelerator working in a performance-oriented profile; on the other hand, DPR enables task-level parallelism by using different hardware accelerators at the same time. The ARTICo^3 runtime library hides all low-level details of FPGA reconfiguration, as well as any parallelism management other than the initial partitioning of the application. A simplified C-based API acts as an interface between user code running on the host processor and the hardware accelerators. As part of this API, functions to manage shared memory buffers, accelerator execution, internal monitors, or additional register-based configuration are also provided.

5. Application Mapping onto ARTICo^3

The initial description of the proposed lossy extension of the CCSDS 123.0-B-1 algorithm was written in pure ANSI C language. After that, modifications were applied to obtain a hardware-friendly description suitable to be implemented on an FPGA using HLS techniques. The main changes are described next.

First of all, the memory architecture was modified to process the samples in BIL instead of BSQ order, as it did initially. The main reason, as it was mentioned in Section 3.3, is that the bit-rate control works in a spectral line, one by one. The design has been partitioned into separate modules that correspond to the different stages of the algorithm. This partitioning allows the verification of the different parts of the algorithm independently. The C-language constructs that are not synthesisable in hardware were redefined, such as dynamic memory allocations or C functions to manage strings. Besides, the use of pointers was modified for avoiding illegal memory accesses.

Taking into account that the target device of this work is a Xilinx Zynq UltraScale+ device, we have exploited the possibilities of applying hardware/software co-design. Functions that are executed a few times every algorithm invocation, such as the header creation or the SB bit-rate control mode, are run in software, using one of the ARM Cortex-A53 cores available in the Processing System (PS). Also, the rate LUT used to determine the suitable quantization step value for the next spectral line is stored in the external RAM. Due to its large size (16-bits integers and a total of 256×1024 elements, supposing 512 kB of memory storage), this approach enables an optimised utilisation of the internal memory elements in the FPGA.

The remaining stages of the compression algorithm, on the other hand, are implemented in hardware using the HLS-based implementation flow of the ARTICo^3 toolchain. In this flow, C-based descriptions are automatically translated into VHDL code using Vivado HLS, and instantiated in a common wrapper with predefined interfaces. Then, a block diagram that contains the processing architecture and the HLS-based accelerators is automatically generated in Vivado. Finally, the Xilinx DPR flow is automatically invoked to create the bitstreams for the static system (i.e., the ARTICo^3 architecture and the bus-based communication infrastructure) and the reconfigurable slots (i.e., the hardware accelerators with part of the compression algorithm). In principle, this approach only needs minor modifications (e.g., replace function calls by proper ARTICo^3 kernel invocations) in the host code (i.e., software functions), while the accelerator code (i.e., hardware functions) remains unmodified. However, the initial algorithm specification required some refinements to enable full compatibility between the accelerator and the ARTICo^3 execution model.

The original compressor code did not support data-level parallelism, since the input hyperspectral image was compressed sequentially, iteratively processing one line at a time and updating the optimal quantification step accordingly. As a result, that solution would not exploit more than one hardware accelerator. To enable data-level parallelism, an image partitioning approach similar to the one implemented in [18] has been developed (i.e., input images are partitioned along both spatial

axes). With the proposed scheme, every hyperspectral image is partitioned in a variable number of fixed-size image segments, which are then independently compressed. The local memory storage in each ARTICo^3-compatible accelerator (i.e., 64 KiB) limits the size of each image segment.

In lossless compression, performing the partitioning mentioned above led to slightly worse compression ratios, but the increase in compression throughput justified its implementation. In the proposed lossy compression algorithm, on the other hand, the effect of image partitioning is negligible in terms of compression ratio, since that parameter is an input of the algorithm. However, the partitioning scheme does affect the quality of the reconstructed images. Figure 6 shows the Signal-to-Noise Ratio (SNR) value of the reconstructed output (i.e., the result of a compression-decompression flow) with different partitioning schemes and target compression ratios. These results have been obtained using a modified AVIRIS image from flight `f080927t01p00r10` with 512 samples, 512 lines and 256 bands. As it can be seen, partitioning over the x-axis (b) generates more considerable differences in the obtained results than partitioning over the y-axis (a) or following a segmentation in square blocks (c), especially when using small target compression ratios. The main reason is that the bit-rate control can get closer to the target bpppb when more spectral lines have been pre-processed and when the length of these lines is enough to perform the median of a different set of L pixels. In addition, the results of the whole prediction stage are also enhanced when the number of pre-processed samples increases, achieving accurate weights updating. Nevertheless, results are acceptable in all compression scenarios, achieving a good trade-off between image quality and compression ratio, even when the image is processed in segments.

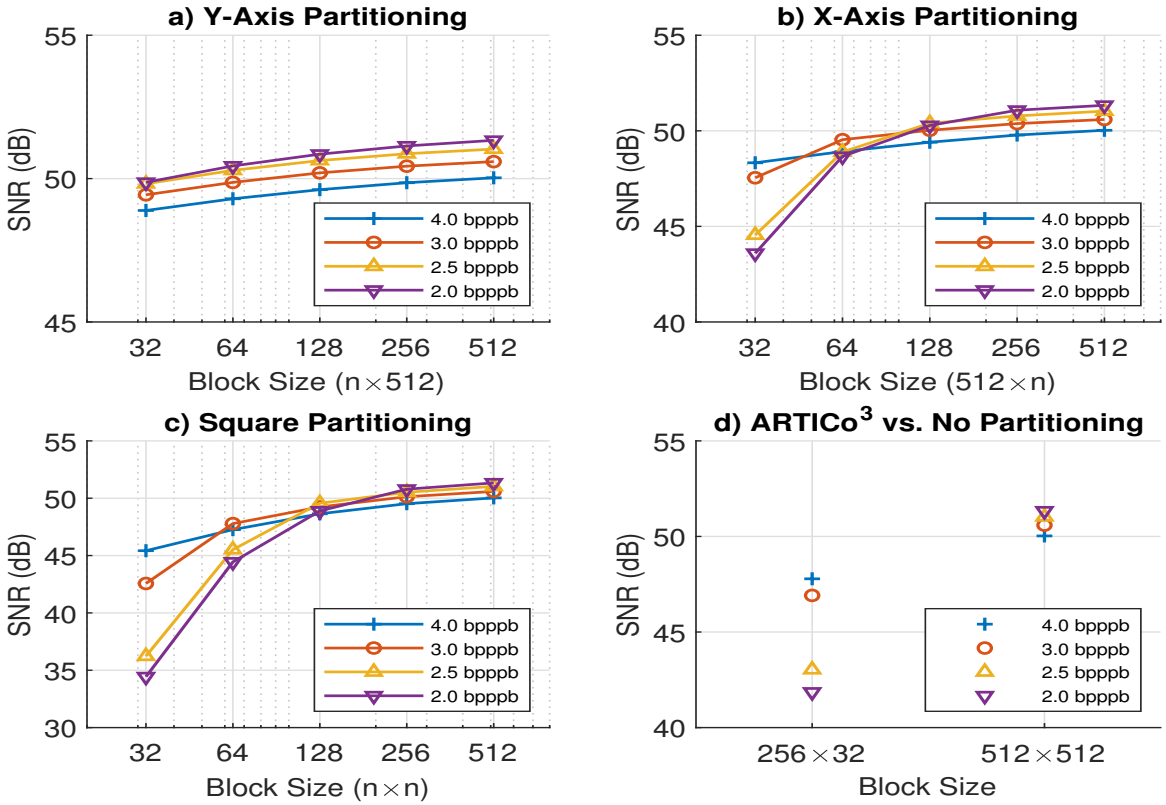

Figure 6. Image partitioning impact on Signal-to-Noise Ratio (SNR) (input versus reconstructed).

In any case, using input image partitioning to support data-level parallelism is not enough to enable full compatibility with ARTICo^3. The fact that the algorithm iteratively processes input segments line by line, and using information from the previously compressed lines, goes against the requirement of data independence between execution rounds in ARTICo^3. Hence, the internal state (e.g., weights) of each hardware accelerator after processing one line of a given segment has to be

saved and immediately restored when the next line of the same segment needs to be processed. This context *save and restore* mechanism has been implemented using some of the local memory banks available inside an ARTICo^3 accelerator.

As a consequence, the local memory map is divided into four different banks: one to store the current input line, two to implement the context save and restore, and one to store the compressed bitstream that is obtained as output. Taking into account the local memory limitation to 64 KiB and that banks are of equal size in each accelerator, a maximum of 16 KiB is available to store input data. This leads to a predefined size of 32 samples and 256 bands per execution round, assuming that two 16-bit integer values are packed in one 32-bit word. The impact of this partitioning approach on the reconstructed image can be seen in Figure 6d.

Further optimizations have been made in the accelerator code by applying static configurations (i.e., predefined parameters for the compressor core, and its prediction and encoder stages at design time) rather than supporting them at run time. Note that this approach does not limit flexibility, since DPR can still be used to change among different compressor configurations, but reduces area overhead by simplifying the internal logic of the accelerators. Table 2 shows the predefined configuration (main parameters) for the lossy compressor core.

Table 2. Main configuration parameters.

Parameter	Value
Samples	32
Lines	256
Bands	256
Dynamic Range	16
Signed	Yes
Endianess	Little Endian
Encoding Order	BIL
Bands for Prediction	3
Local Sum Mode	Neighbour-Oriented
Prediction Mode	Full Prediction

Finally, control and status parameters are sent to or received from the accelerators using the register-based interface featured by the ARTICo^3 architecture. Up to 8 32-bit registers are used to store information such as the index of the current line, or the quantization step size.

6. Experimental Results

6.1. Bit-Rate Control

First of all, the SB bit-rate control mode is compared against modes A and B detailed in Section 3.3.2, to analyse its accuracy for obtaining the targeted bit-rate. Besides, the quality of the reconstructed image after the decompression stage is measured in terms of PSNR, taking into account the bit-rate control used. The results are summarized in Table 3.

As it can be observed, the SB mode achieves similar results than the B one in terms of accuracy to reach the targeted bit-rate, obtaining a maximum error of 0.023 bpppb when targeting 4 bpppb. Regarding the PSNR results, the selection of the best bit-rate control mode is dependent on the target bit-rate and the image nature. In this sense, the best results for AVIRIS images are obtained by the SB mode targeting 2 bpppb; otherwise, the B mode achieves higher PSNR, with a maximum difference of around 3.2 dB targeting 4 bpppb. Nevertheless, taking into account the higher computational complexity of mode B, the SB mode can be considered as a trade-off between the quality of the obtained results and complexity. In general, the A mode provides the worst results in terms of both achieved bpppb and PSNR. This is due to the absence of feedback from the entropy coding stage,

indicating the differences between the targeted and the reached bit-rate, later used to adapt the quantization step for compressing the next spectral line.

Table 3. Bit-rate accuracy depending on the selected control mode.

Targeted Bit-Rate	Mode	Reached Bit-Rate (bpppb)	PSNR (dB)
2 bpppb	A	2.268	80.610
	B	2.003	73.762
	SB	2.011	76.571
3 bpppb	A	2.475	82.218
	B	3.001	85.533
	SB	2.999	84.618
4 bpppb	A	2.632	83.358
	B	4.011	91.823
	SB	3.977	88.589

6.2. Resource Utilization and Performance Results on ARTICo³

The proposed implementation of the lossy hyperspectral compressor has been evaluated on a Xilinx ZCU102 development board, which features a Zynq UltraScale+ device (XCZU9EG-2FFVB900). A template with 8 reconfigurable slots has been created to support that specific device in the ARTICo³ toolchain. The layout of the implemented system can be seen in Figure 7, with the static region highlighted in orange and the reconfigurable accelerators in blue. Each reconfigurable region is highlighted in red. Table 4 shows the resources utilization report for the ARTICo³ infrastructure (i.e., static region) and one instance of the lossy compression kernel after implementing the multi-accelerator system with the predefined configuration parameters, as discussed in the previous section. Relative utilization values (i.e., the percentage of used versus total resources available in the target device) are also provided.

Table 4. Resource utilization.

Component	ARTICo³	Lossy Compressor Kernel
Info	8 slots VHDL (Vivado) Zynq UltraScale+	64 KiB memory, 4 banks 8 registers C + HLS (Vivado) Zynq UltraScale+
LUTs	5560 (2.03%)	6405 (2.34%)
FFs	3763 (0.69%)	4876 (0.89%)
DSPs	2 (0.08%)	3 (0.12%)
BRAMs	-	19.5 (2.14%)

Figure 8 shows the execution time of the lossy hyperspectral compressor when changing the number of hardware accelerators (quad-core ARM Cortex-A53 @ 1.2 GHz, FPGA fabric @ 100 MHz). The host application code runs on a Linux-based OS and compresses hyperspectral images of 512 samples, 512 lines, and 256 bands (a modified AVIRIS image from flight `f080927t01p00r10`.). The host application divides the input image into several segments of 32 samples, 256 lines, and 256 bands, and uses the pool of available hardware accelerators to iteratively compress all the image segments one line at a time. This partition scheme should be taken into account on the decompression side, to properly join the different fractions for recovering the whole decompressed image.

Experimental results show execution times of 174.5 and 35.92 s using 1 and 8 ARTICo³ accelerators, respectively. Taking into account that the reference software implementation (i.e., initial C code without any acceleration libraries) running as a single-core application on an ARM Cortex-A53 @ 1.2 GHz takes roughly 560 s to process the same input image, the multi-accelerator system provides speedups of up

to 15.6×. Notice, however, that the performance scalability is not entirely linear (i.e., the speedup is roughly 5× when using 8 hardware accelerators instead of only one).

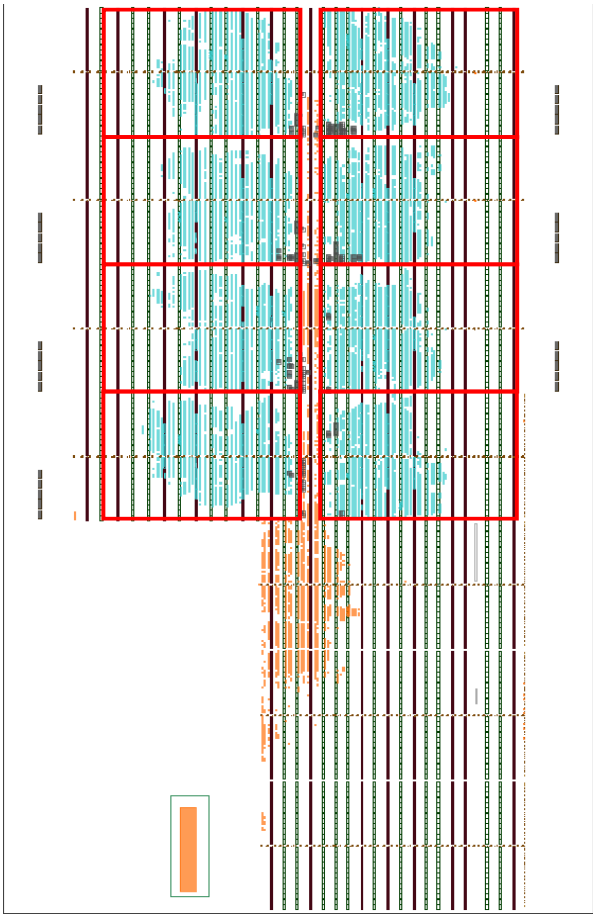

Figure 7. Layout of the implemented system on the ZCU102 development board.

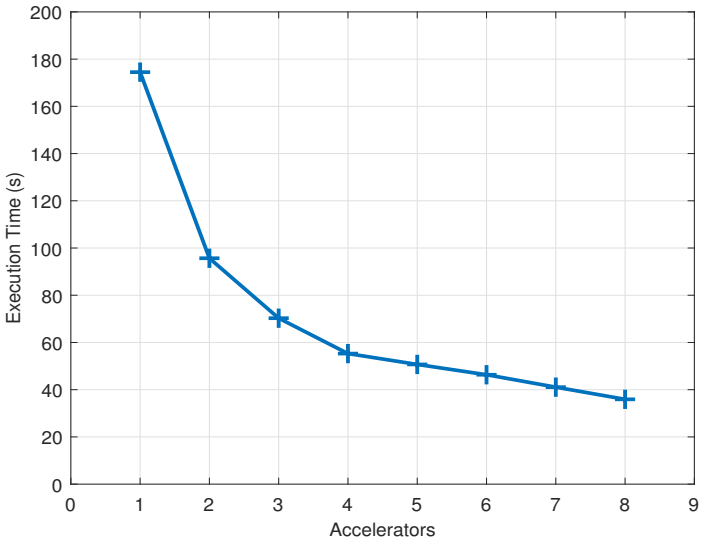

Figure 8. Performance versus number of accelerators.

Figure 9 shows the execution time of the lossy hyperspectral compressor when changing the size of the input images. This setup uses the same input image and operating frequencies from the previous test, but a variable number of segments (each of them with the fixed size of 32 samples, 256 lines,

and 256 bands imposed by ARTICo^3) is fed to the compression system. In addition, the number of accelerators is modified accordingly to enable maximum parallelism during the processing stage.

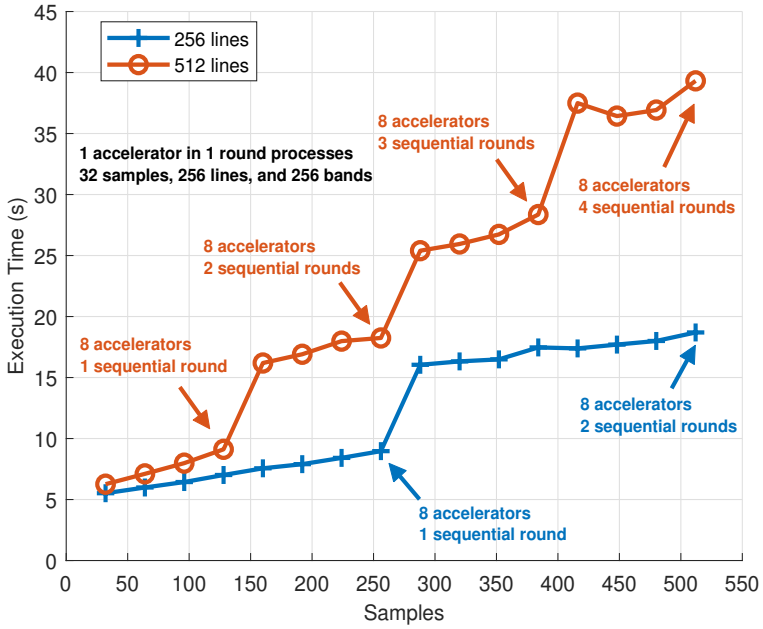

Figure 9. Performance versus image size.

A linear evolution is observed in the execution times when the number of hardware accelerators loaded in the FPGA is enough to allow parallel compression of different image segments. Moreover, sharp transitions appear when the previous condition is not met, and the compression of different image segments needs to be serialized. Once sequential processing is forced, the system shows linear behaviour again until extra serial iterations are required. This particular evolution can be seen in the following example: execution time increases linearly when compressing images with 256 lines whose samples range from 32 to 256 (i.e., 1 to 8 segments with up to 8 accelerators) and when compressing images with 256 lines whose samples range from 288 to 512 (i.e., 9 to 16 segments with 8 accelerators), with an abrupt change in between due to the number of iterations going from 1 to 2. Notice that the small variations in the graph are due to the non-deterministic nature of the Linux-based OS, although results are averaged over several executions.

6.3. Analysis of the Compression Ratio

Finally, to verify the results in terms of achieved compression ratio, the proposed solution was validated with hyperspectral images coming from different types of sensors, such as the AVIRIS mentioned above (aircraft), the Compact Reconnaissance Imaging Spectrometer for Mars (CRISM) and Landsat spacecrafts, and a subset of an image captured directly by a Headwall Hyperspec VNIR E-Series hyperspectral camera, available in our hyperspectral image acquisition facilities. Table 5 summarizes the main features of these images. This way, the implementation is tested in different real-world scenarios varying the number of spectral components, from multispectral to ultraspectral images.

The compression ratio achieved for each one of the targeted images in terms of bpppb is shown in Figure 10, against the quality of the reconstructed image after the decompression stage. The quality of the image is measured using the SNR, obtaining higher values (a maximum of 90 dB for the CRISM sensor) when the images have undergone any kind of pre-processing (e.g., calibration) after its acquisition and before being compressed. Results reported for the Landsat image demonstrate that the behaviour of the lossy compression algorithm is improved when it has more spectral information for performing the statistical operations (i.e., the median calculation). In the case of the AVIRIS

scene, results in terms of image quality are in line with the ones provided by the CCSDS 123.0-B-2 algorithm using different prediction configurations, as it is reported in [31]. In addition, the proposed implementation overcomes the SNR results provided not only by the CCSDS 122.1-B-1 standard, based on the use of a spectral transform for reducing the decorrelation between adjacent bands, but also by the well-known JPEG2000 standard, reducing at the same time the implementation complexity. For the particular case of the AVIRIS dataset, our proposal achieves better results than both the IWT and the POT alternatives, targeting compression ratios higher than 2 bpppb [49]. Notice that the maximum compression ratio achieved by our work (i.e., 2 bpppb) is a constraint introduced by both the selected entropy coder, based on a sample-adaptive scheme, and the maximum quantization step size (i.e., 511). Higher compression ratios can be obtained using a rate LUT with higher quantization step values or implementing alternative encoding options, such as the new hybrid encoder presented in [33].

Table 5. Images to validate the algorithm implementation.

Image	Nx × Ny × Nz	D	Signed	Endianness	State
AVIRIS Yellowstone sc0	680×512×224	16	No	BIG	Processed
CRISM scn164VNIR	640×420×107	12	No	LITTLE	Calibrated
Landsat Mountain	1024×1024×6	8	No	BIG	Raw
Hyperspec VNIR E-Series	400×400×300	16	Yes	LITTLE	Raw

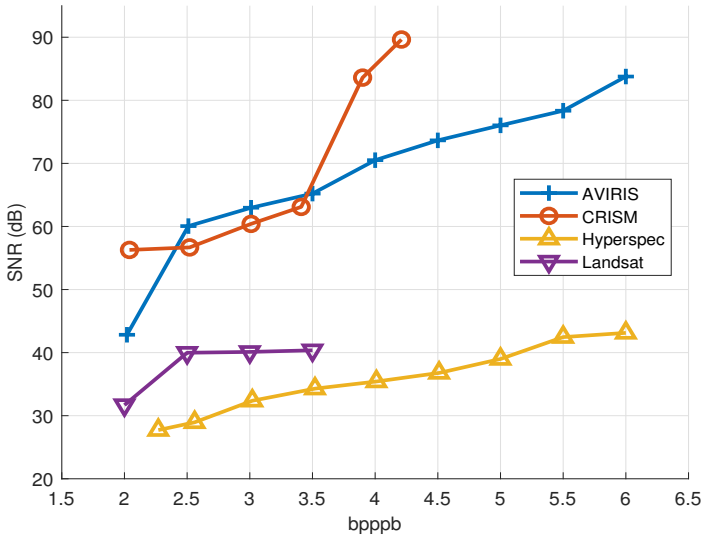

Figure 10. Compression ratio versus SNR for different sensor images.

In all the analysed cases, the design reaches the targeted compression ratio specified by the user with accuracy. Notice that beyond the limits defined for each image in Figure 10, the compressor works in lossless mode.

6.4. Comparison with Other Implementations

In this section, results obtained for the proposed architecture are compared with previous FPGA implementations available in the state-of-the-art. The comparison is limited to implementations of near-lossless and lossy compression algorithms, since lossless solutions are generally less complex at the expense of lower compression ratios. It is worth noting that the diversity of algorithms in the

selected works and the fact that different FPGA technologies are used, make it necessary to include qualitative parameters, apart from quantitative results, for a fair comparison.

The implementations analyzed are summarized in Table 6, including both performance results and resource utilization metrics for each one of them, when available. A fair comparison in terms of throughput is provided, considering samples with a precision of 16 bits for all the cases.

Table 6. Comparison with other Field-Programmable Gate Array (FPGA) implementations of lossy compression solutions.

Implementation	Device	Run-Time Adaptability	Bit-Rate Control	CCSDS123 Based	LUTs	FFs	DSPs	BRAMs	Freq. (MHz)	Throughput (MSamples/s)
Keymulen (1 core) [35]	XC7VX690T	No	Yes	Yes	13,134	9158	31	36	200	9
Keymulen (15 cores) [35]	XC7VX690T	No	Yes	Yes	297,807	260,750	465	556	200	95
Abrardo et al. [50]	XQR4VLX200	No	No	No	10,306	-	9	21	81	70
Santos et al. [38]	XQR5VFX130	No	No	No	7836	4208	4	17	80.2	30.25
García et al. [51]	XC5VFX100	No	No	No	7746	-	25	4	86	27.7
Báscones et al. [40]	XQR5VFX130	No	No	No	6837	-	5	10	247.35	119.96
This work (1 core)	XCZU9EG	Yes	Yes	Yes	11,965	8639	5	19.5	100	0.4
This work (8 cores)	XCZU9EG	Yes	Yes	Yes	56,800	42,771	26	156	100	1.7

The implementation of the FLEX algorithm proposed by Keymulen [35] introduces a feedback branch with an inverse predictor, as defined in the CCSDS 123.0-B-2 standard. In this way, it is possible to accurately estimate the value of the subsequent samples without compromising the image quality, which is controlled by a maximum error defined by the user. This architecture can achieve up to 95 MSamples/s when 15 cores are working simultaneously in a Xilinx Virtex-7 (XC7VX690T), even though it also shows high area occupation. This architecture has been successfully validated on-board in the ECOsystem Spaceborne Thermal Radiometer Experiment on Space Station (ECOSTRESS), which acquires images with 6 spectral components and it is currently at the ISS for scientific data collection. In this scenario, the Digital Processing Unit (DPU) is a radiation-hardened Virtex-5 (XQR5VFX130), and the compressor works at 100 MHz, reaching a throughput of 1 MSamples/s [34]. Although this solution is scalable, the addition of parallel compression instances to increase the throughput is done neither at run-time nor taking into account other necessities identified during the algorithm execution, as it is done in the proposed ARTICo^3-based implementation.

The rest of the hardware implementations taken into account in this comparison are based on the prediction scheme detailed in [28]. This algorithm is based on the independent compression of small image blocks in the spatial domain, named slices, with all the spectral components. This division allows the parallelisation of the compression process because different slices can be compressed simultaneously. However, a small penalty is observed in terms of performance, since the state of the different compression statistics has to be reset at the end of each slice. The work in [28] introduces a Uniform-Threshold Quantizer (UTQ), instead of the scalar uniform quantizer employed in our implementation. With this change, better performance is obtained for high bit-rates. Nevertheless, this algorithm does not include a true bit-rate control; it only defines a threshold to guide the compression process and to skip the encoding of the next slices if the prediction is considered "good enough", writing the prediction error in the output bitstream directly. This algorithmic trait prevents an accurate control of the rate-distortion figure. Our implementation, on the contrary, does enable this precise control, but at the expense of an increased resource utilisation and a reduced throughput.

Four different FPGA implementations of this predictive algorithm are considered in the comparison: two described in VHDL [40,50], and the other two using HLS techniques [38,51]. In this case, it is observed that there are not too many differences regarding resource utilisation (typically, an HLS implementation employs 3× the resources consumed by the equivalent application described using VHDL), even though better results in terms of throughput are obtained by the VHDL implementations, which in turn allow a more precise scheduling of the algorithm operations. Concretely, the best timing results are obtained by the implementation proposed in [40], a highly

optimized solution that reaches a maximum throughput of almost 120 MSamples/s when it is mapped onto a radiation-hardened Virtex-5 (XQR5VFX130).

Quantitatively speaking, the throughput of our proposal is limited when compared with the rest of the works shown in Table 6. These differences were foreseeable for the implementations of the predictive algorithm described in [28], since the associated compressor architectures do not include a feedback loop to control both image quality and target bpppb. The work proposed by Keymulen et al. [35], on the other hand, does feature a feedback branch in the predictor. However, and although the resource utilisation values are in the same order of magnitude, there are still differences in terms of throughput. There are two main reasons that explain these differences: the first one is that the hardware implementation of the proposed compression solution is not fully optimized (i.e., there is still room for improvement by applying HLS-based optimization directives in the hardware accelerator code), since algorithm-wise modifications (e.g., achieving the target bpppb value without incurring a significant degradation of the image quality) were prioritised over implementation-wise modifications (e.g., loop pipelining or unrolling); the second one is that the hardware/software partitioning of the implementation still exhibits inefficient parts (e.g., excessive accelerator state save and restore operations).

However, the key contributions of the proposed ARTICo^3-based lossy compressor need to be evaluated qualitatively. In this regard, two aspects can be highlighted: first, that our proposal is based on the CCSDS-123 standard solution (as it is the work reported by Keymulen et al. [35]), but modified to reach specific compression ratios; second, that our proposal is the only solution also capable of providing run-time adaptive behaviour. The reconfigurable multi-accelerator substrate of the ARTICo^3 architecture, together with its runtime management infrastructure, enables user-selectable performance (i.e., load more hardware accelerators to transparently increase throughput), power consumption, and fault tolerance (i.e., group hardware accelerators to transparently enable hardware redundancy). The fact that it is implemented on top of COTS FPGA devices makes it a suitable solution for low-cost scientific missions under extremely changing conditions, both external (e.g., solar storms require increased fault tolerance in the system) and internal (e.g., low battery levels require energy-efficient operation modes). Moreover, the DPR support in ARTICo^3 allows different applications to coexist within the same FPGA device, which effectively reduces cost (e.g., two mutually exclusive algorithms can be time-multiplexed on the hardware resources and thus, smaller FPGAs can be used without incurring actual performance losses) and enables system updates after deployment (e.g., send new processing cores from Earth to update the FPGA bitstream).

7. Conclusions and Future Work

In this paper, a lossy extension of the CCSDS 123.0-B-1 Lossless Multispectral and Hyperspectral Image Compression algorithm is presented. This solution is implemented on a Xilinx Zynq UltraScale+ FPGA-based MPSoC, running on ARTICo^3, a reconfigurable, scalable and fault-tolerant computing architecture.

According to the obtained results, we conclude that the proposed implementation provides an efficient and flexible solution for on-board processing applications without compromising the compression quality. The intrinsic benefits of using the ARTICo^3 architecture as the main computing platform enable run-time adaptive performance and fault tolerance levels, a must-have feature in space missions with changing requirements (e.g., communication link quality in alternative points of a satellite orbit might be different). Moreover, its associated DPR mechanisms, which can be used to change not only the number but also the type of hardware accelerators, favour reduced resource utilisation values by time-multiplexing the FPGA fabric. The system achieves a maximum speed up of 15.6$\times$ when the maximum number of available accelerators is used, in comparison with the pure software version of the algorithm running on an ARM Cortex-A53. However, there is still a margin to improve the throughput of the design. Therefore, the next step is to perform an exhaustive study of the algorithm loops to be able to exploit the parallel nature of FPGAs, applying techniques such as

unrolling or pipelining, if possible, without incurring an excessive increment of the resource utilisation (Figure 7 shows that there is still space available in the reconfigurable slots).

In addition, this implementation will be evaluated against radiation effects using a fault-injection engine, being able to emulate the conditions of the space environment and reducing time and costs in comparison with traditional beam campaigns. In this way, it is possible to characterise the design and identifying specific parts of the logic that must be hardened. For this purpose, the fault-management engine relies on the Xilinx Soft Error Mitigation (SEM) IP [52] both to inject Single Event Upsets (SEUs) and to perform SEU detection and correction, followed by a classification of the faults present in the configuration memory (scrubbing tasks) [53]. The dual-core ARM Cortex-R5 available in the Zynq UltraScale+ device will be used to manage the execution of this tool and to monitor the results of the fault-injection process, respectively.

Author Contributions: Conceptualization, S.L. and A.O.; methodology, Y.B. and A.R.; investigation, Y.B., A.R. and A.S.; validation, Y.B., A.R. and A.P.; software, A.S. and A.P.; writing—original draft preparation, Y.B. and A.R.; writing—review and editing, Y.B., A.R., A.S. and A.O.; supervision, S.L. and A.O.; project administration, R.S. and E.d.l.T. All authors have read and agreed to the published version of the manuscript.

Funding: This work has been conducted within the ENABLE-S3 project that has received funding from the ECSEL Joint Undertaking under grant agreement No 692455. This Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation program and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France,Netherlands, United Kingdom, Slovakia, Norway. This work was also partially supported by the Spanish Ministry of Economy and Competitiveness under the project PLATINO, with reference numbers TEC2017-86722-C4-1-R and TEC2017-86722-C4-2-R.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Staenz, K.; Mueller, A.; Heiden, U. Overview of terrestrial imaging spectroscopy missions. In Proceedings of the 2013 IEEE International Geoscience and Remote Sensing Symposium—IGARSS, Melbourne, VIC, Australia, 21–26 July 2013; pp. 3502–3505. [CrossRef]
2. Transon, J.; d'Andrimont, R.; Maugnard, A.; Defourny, P. Survey of Hyperspectral Earth Observation Applications from Space in the Sentinel-2 Context. *Remote Sens.* **2018**, *10*, 157. [CrossRef]
3. Christophe, E. Hyperspectral Data Compression Tradeoff. In *Optical Remote Sensing, Advances in Signal Processing and Exploitation Techniques*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 9–29.
4. George, A.D.; Wilson, C.M. Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites. *Proc. IEEE* **2018**, *106*, 458–470. [CrossRef]
5. Wirthlin, M. High-Reliability FPGA-Based Systems: Space, High-Energy Physics, and Beyond. *Proc. IEEE* **2015**, *103*, 379–389. [CrossRef]
6. Zhu, J.; Dutt, N. Electronic System-Level design and High-Level Synthesis. In *Electronic Design Automation: Synthesis, Verification, and Test*; Elsevier: Amsterdam, The Netherlands, 2009; pp. 235–298.
7. The Consultative Committee for Space Data Systems. *Lossless Multispectral and Hyperspectral Image Compression, CCSDS 123.0-B-1*; Blue Book ed.; CCSDS: Reston, VA, USA, 2012; Volume 1.
8. Rodríguez, A.; Valverde, J.; Portilla, J.; Otero, A.; Riesgo, T.; de la Torre, E. FPGA-Based High-Performance Embedded Systems for Adaptive Edge Computing in Cyber-Physical Systems: The ARTICo^3 Framework. *Sensors* **2018**, *18*, 1877. [CrossRef] [PubMed]
9. Barrios, Y.; Sánchez-Clemente, A.J.; Sarmiento, R.; Rodríguez, A.; Otero, A.; de la Torre, E. Hyperspectral Image Lossy Compression on a Reconfigurable and Fault-Tolerant Architecture Implemented over a COTS FPGA-Based System-on-Chip. In Proceedings of the 6th International Workshop on On-Board Payload Data Compression (OBPDC), Matera, Italy, 20–21 September 2018; European Space Agency: Paris, France, 2018; pp. 1–7.
10. Santos, L.; Berrojo, L.; Moreno, J.; Lopez, J.F.; Sarmiento, R. Multispectral and Hyperspectral Lossless Compressor for Space Applications (HyLoC): A Low-Complexity FPGA Implementation of the CCSDS 123 Standard. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2016**, *9*, 757–770. [CrossRef]

11. Keymeulen, D.; Aranki, N.; Bakhshi, A.; Luong, H.; Sarture, C.; Dolman, D. Airborne demonstration of FPGA implementation of Fast Lossless hyperspectral data compression system. In Proceedings of the 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Leicester, UK, 14–18 July 2014; pp. 278–284. [CrossRef]
12. Tsigkanos, A.; Kranitis, N.; Theodorou, G.A.; Paschalis, A. A 3.3 Gbps CCSDS 123.0-B-1 Multispectral Hyperspectral Image Compression Hardware Accelerator on a Space-Grade SRAM FPGA. *IEEE Trans. Emerg. Top. Comput.* **2019**, 1. [CrossRef]
13. Santos, L.; Gomez, A.; Sarmiento, R. Implementation of CCSDS Standards for Lossless Multispectral and Hyperspectral Satellite Image Compression. *IEEE Trans. Aerosp. Electron. Syst.* **2019**, 56, 1120–1138. [CrossRef]
14. Barrios, Y.; Sánchez, A.; Santos, L.; Sarmiento, R. SHyLoC 2.0: A versatile hardware solution for on-board data and hyperspectral image compression on future space missions. *IEEE Access* **2020**, *8*, 54269–54287. [CrossRef]
15. Bascones, D.; Gonzalez, C.; Mozos, D. FPGA Implementation of the CCSDS 1.2.3 Standard for Real-Time Hyperspectral Lossless Compression. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2018**, *11*, 1158–1165. [CrossRef]
16. Pereira, L.M.V.; Santos, D.A.; Zeferino, C.A.; Melo, D.R. A Low-Cost Hardware Accelerator for CCSDS 123 Predictor in FPGA. In Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, 26–29 May 2019; pp. 1–5. [CrossRef]
17. Fjeldtvedt, J.; Orlandic, M.; Johansen, T.A. An Efficient Real-Time FPGA Implementation of the CCSDS-123 Compression Standard for Hyperspectral Images. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2018**, *11*, 3841–3852. [CrossRef]
18. Rodríguez, A.; Santos, L.; Sarmiento, R.; de la Torre, E. Scalable Hardware-Based On-Board Processing for Run-Time Adaptive Lossless Hyperspectral Compression. *IEEE Access* **2019**, *7*, 10644–10652. [CrossRef]
19. Bascones, D.; Gonzalez, C.; Mozos, D. Parallel Implementation of the CCSDS 1.2.3 Standard for Hyperspectral Lossless Compression. *Remote Sens.* **2017**, *9*, 973. [CrossRef]
20. Orlandic, M.; Fjeldtvedt, J.; Johansen, T.A. A Parallel FPGA Implementation of the CCSDS-123 Compression Algorithm. *Remote Sens.* **2019**, *11*, 673. [CrossRef]
21. Davidson, R.L.; Bridges, C.P. GPU accelerated multispectral EO imagery optimised CCSDS-123 lossless compression implementation. In Proceedings of the 2017 IEEE Aerospace Conference, Big Sky, MT, USA, 4–11 March 2017; pp. 1–12. [CrossRef]
22. Hopson, B.; Benkrid, K.; Keymeulen, D.; Aranki, N. Real-time CCSDS lossless adaptive hyperspectral image compression on parallel GPGPU multicore processor systems. In Proceedings of the 2012 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Erlangen, Germany, 25–28 June 2012; pp. 107–114. [CrossRef]
23. Penna, B.; Tillo, T.; Magli, E.; Olmo, G. Progressive 3-D coding of hyperspectral images based on JPEG 2000. *IEEE Geosci. Remote Sens. Lett.* **2006**, *3*, 125–129. [CrossRef]
24. Egho, C.; Vladimirova, T. Adaptive hyperspectral image compression using the KLT and integer KLT algorithms. In Proceedings of the 2014 NASA/ESA Conference on Adaptive Hardware and Systems (AHS), Leicester, UK, 14–17 July 2014; pp. 112–119.
25. Penna, B.; Tillo, T.; Magli, E.; Olmo, G. Transform Coding Techniques for Lossy Hyperspectral Data Compression. *IEEE Trans. Geosci. Remote Sens.* **2007**, *45*, 1408–1421. [CrossRef]
26. Blanes, I.; Serra-Sagristà, J. Pairwise Orthogonal Transform for Spectral Image Coding. *IEEE Trans. Geosci. Remote Sens.* **2011**, *49*, 961–972. [CrossRef]
27. The Consultative Committee for Space Data Systems. *Spectral Preprocessing Transform for Multispectral and Hyperspectral Image Compression, CCSDS 122.1-B-1*; Blue Book ed.; CCSDS: Washington, DC, USA, 2017; Volume 1.
28. Abrardo, A.; Barni, M.; Magli, E. Low-complexity predictive lossy compression of hyperspectral and ultraspectral images. In Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Prague, Czech Republic, 22–27 May 2011; pp. 797–800.
29. Guerra, R.; Barrios, Y.; Díaz, M.; Santos, L.; López, S.; Sarmiento, R. A New Algorithm for the On-Board Compression of Hyperspectral Images. *Remote Sens.* **2018**, *10*, 428. [CrossRef]

30. Penna, B.; Tillo, T.; Magli, E.; Olmo, G. Hyperspectral Image Compression Employing a Model of Anomalous Pixels. *IEEE Geosci. Remote Sens. Lett.* **2007**, *4*, 664–668. [CrossRef]
31. Valsesia, D.; Magli, E. High-Throughput Onboard Hyperspectral Image Compression With Ground-Based CNN Reconstruction. *IEEE Trans. Geosci. Remote Sens.* **2019**, *57*, 9544–9553. [CrossRef]
32. Báscones, D.; González, C.; Mozos, D. Hyperspectral Image Compression Using Vector Quantization, PCA and JPEG2000. *Remote Sens.* **2018**, *10*, 907. [CrossRef]
33. The Consultative Committee for Space Data Systems. *Low-Complexity Lossless and Near-Lossless Multispectral and Hyperspectral Image Compression, CCSDS 123.0-B-2*; Blue Book ed.; CCSDS: Washington, DC, USA, 2019; Volume 1.
34. Keymeulen, D.; Dolman, D.; Shin, S.; Riddley, J.; Klimesh, M.; Kiely, A.; Thompson, D.R.; Cheng, M.; Dolinar, S.; Liggett, E.; et al. High Performance Space Data Acquisition, Clouds Screening and Data Compression with modified COTS Embedded System-on-Chip Instrument Avionics for Space-based Next Generation Imaging Spectrometers (NGIS). In Proceedings of the 6th International Workshop on On-Board Payload Data Compression (OBPDC), Matera, Italy, 20–21 September 2018; European Space Agency: Paris, France, 2018; pp. 1–25.
35. Keymeulen, D. FPGA Implementation of Lossless and Lossy Compression of Space-based Multispectral and Hyperspectral Imagery. In Proceedings of the Military and Aerospace Programmable Logic Devices (MAPLD) Workshop, La Jolla, CA, USA, 21–24 May 2018; Jet Propulsion Laboratory, National Aeronautics and Space Administration: Pasadena, CA, USA, 2016; pp. 1–28.
36. Santos, L.; Blanes, I.; García, A.; Serra-Sagristà, J.; López, J.; Sarmiento, R. On the hardware implementation of the arithmetic elements of the pairwise orthogonal transform. *J. Appl. Remote Sens.* **2015**, *9*, 097496. [CrossRef]
37. Guerra, R.; Barrios, Y.; Díaz, M.; Baez, A.; López, S.; Sarmiento, R. A Hardware-Friendly Hyperspectral Lossy Compressor for Next-Generation Space-Grade Field Programmable Gate Arrays. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2019**, *12*, 4813–4828. [CrossRef]
38. Santos, L.; López, J.F.; Sarmiento, R.; Vitulli, R. FPGA implementation of a lossy compression algorithm for hyperspectral images with a high-level synthesis tool. In Proceedings of the 2013 NASA/ESA Conference on Adaptive Hardware and Systems (AHS-2013), Torino, Italy, 24–27 June 2013; pp. 107–114.
39. García, A.; Santos, L.; López, S.; Callicó, G.M.; López, J.F.; Sarmiento, R. FPGA implementation of the hyperspectral Lossy Compression for Exomars (LCE) algorihtm. In *High-Performance Computing in Remote Sensing IV*; Huang, B., López, S., Wu, Z., Eds.; International Society for Optics and Photonics, SPIE: Bellingham, WA, USA, 2014; Volume 9247, pp. 27–34. [CrossRef]
40. Báscones, D.; González, C.; Mozos, D. An Extremely Pipelined FPGA Implementation of a Lossy Hyperspectral Image Compression Algorithm. *IEEE Trans. Geosci. Remote Sens.* **2020**, 58, 7435–7447. [CrossRef]
41. Santos, L.; Magli, E.; Vitulli, R.; López, J.F.; Sarmiento, R. Highly-Parallel GPU Architecture for Lossy Hyperspectral Image Compression. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2013**, *6*, 670–681. [CrossRef]
42. Díaz, M.; Guerra, R.; Horstrand, P.; Martel, E.; López, S.; López, J.F.; Sarmiento, R. Real-Time Hyperspectral Image Compression Onto Embedded GPUs. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2019**, *12*, 2792–2809. [CrossRef]
43. ESA. SHyLoC IP Core. Available online: https://www.esa.int/Our_Activities/Space_Engineering_Technology/Microelectronics/SHyLoC_IP_Core (accessed on 14 September 2018).
44. Augé, S.; Sánchez, J.E.; Kiely, A.; Blanes Garcia, I.; Serra Sagrisà, J. Performance impact of parameter tuning on the CCSDS-123 lossless multi- and hyperspectral image compression standard. *J. Appl. Remote Sens.* **2013**, *7*, 074594. [CrossRef]
45. The Consultative Committee for Space Data Systems. *Lossless Data Compression, CCSDS 121.0-B-2*; Blue Book ed.; CCSDS: Washington, DC, USA, 2012; Volume 1.
46. CCSDS. *Lossless Multispectral and Hyperspectral Image Compression, Informational Report CCSDS 120.2-G-1*; Green Book; CCSDS: Washington, DC, USA, 2015.
47. Valsesia, D.; Magli, E. A Novel Rate Control Algorithm for Onboard Predictive Coding of Multispectral and Hyperspectral Images. *IEEE Trans. Geosci. Remote Sens.* **2014**, *52*, 6341–6355. [CrossRef]

48. Valsesia, D.; Magli, E. Fast and Lightweight Rate Control for Onboard Predictive Coding of Hyperspectral Images. *IEEE Geosci. Remote Sens. Lett.* **2017**, *14*, 394–398. [CrossRef]
49. CCSDS. *Spectral Pre-Processing Transform for Multispectral & Hyperspectral Image Compression, Informational Report CCSDS 120.3-G-1*; Green Book; CCSDS: Washington, DC, USA, 2019.
50. Abrardo, A.; Barni, M.; Bertoli, A.; Grimoldi, R.; Magli, E.; Vitulli, R. Low-Complexity Approaches for Lossless and Near-Lossless Hyperspectral Image Compression. In *Satellite Data Compression*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 47–65.
51. García, A.; Santos, L.; López, S.; Marrero, G.; López, J.F.; Sarmiento, R. High level modular implementation of a lossy hyperspectral image compression algorithm on a FPGA. In Proceedings of the 2013 5th Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), Gainesville, FL, USA, 26–28 June 2013; pp. 1–4.
52. Xilinx Inc. Soft Error Mitigation Controller v4.1. In *LogiCORE IP Product Guide*; Xilinx: San Jose, CA, USA, 2018.
53. Pérez, A.; Rodríguez, A.; Otero, A.; Arjona, D.G.; Jiménez-Peralo, Á.; Verdugo, M.A.; De La Torre, E. Run-Time Reconfigurable MPSoC-Based On-Board Processor for Vision-Based Space Navigation. *IEEE Access* **2020**, *8*, 59891–59905. [CrossRef]