FPGA-Based Hyperspectral Lossy Compressor With Adaptive Distortion Feature for Unexpected Scenarios

Julián Caba, Dirk Stroobandt, *Member, IEEE*, María Díaz, Jesús Barba, Fernando Rincón, *Member, IEEE*, Sebastián López, *Member, IEEE*, and Juan Carlos López, *Member, IEEE*

***Abstract*—Lossy compression solutions have grown up during the past decades because of the increment of the data rate in the new-generation hyperspectral sensors; however, linear compression techniques include useless information on regions of little interest for the final application and, at the same time, scarce information on areas of interest. In this article, a transform-based lossy compressor, HyperLCA, has been extended to include a run-time adaptive distortion feature that brings multiple compression ratios in the same scenario. The solution has been designed to keep the same hardware-friendly feature, just as its previous version, specifically conceived to ease the deployment of the solution on reconfigurable hardware devices (FPGAs). The experiments demonstrate that the new version of the compressor is able to process 1024 $\times$ 1024 hyperspectral images and 180 spectral bands (377.5 MB) in 0.935 s with a power consumption of 1.145 W. In addition, experimental results also reveal that our architecture features high throughput (MSamples/s) and remarkable energy-efficiency (MB/s/W) tradeoffs, 10$\times$ and 6$\times$ greater than the best state-of-the-art solution, respectively.**

***Index Terms*—Adaptive computing, field-programmable gate array (FPGA), hyperspectral imaging, lossy compression, on-board processing.**

I. Introduction

THE information collected by hyperspectral sensors in the electromagnetic spectrum provides richness of spectral information, especially if they are high resolution and multichannel, to be processed by applications related to the Earth's observation. This fact makes hyperspectral technology a leading candidate for the analysis of land areas and, hence, has acquired an important relevance, being widely used for a variety of remote sensing applications, such as precision agriculture, geological mapping, or mineral exploration. Nevertheless, the large amount of data collected by these sensors requires huge on-board storage resources or high-bandwidth communications, but both are limited. On top of that, the technological advances promote to market sensors with higher spectral and spatial resolutions that make on-board data processing more challenging [1], [2].

Traditionally, the pieces of information captured by hyperspectral sensors are not processed onboard due to the low computing performance and the limited on-board power capacity. Thus, low-power and cost-optimized devices are selected to process the data sensed, but these devices do not feature a high performance [3]. In this regard, images are downloaded to the Earth's surface to be processed ofline by high-performance computing systems. In aerial capturing platforms, such as unmanned aerial vehicles (UAVs), images are usually stored onboard and processed when the flight mission is completed [4]. Recently, some efforts have been made to transmit the images to the ground as soon as they are captured, but it requires a point-to-point connection with high bandwidth in order to reduce large delays [5]. Consequently, the transfer of large volumes of data reveals a bottleneck in the downlink systems that can affect the overall performance, as well as the budget of energy consumed by the transmission [6].

Recent studies propose edge computing solutions that allow to process the sensed images onboard by reducing the amount of downlink bandwidth. Most hyperspectral imaging applications require the processing of huge data using complex algorithms with a formidable computational burden [7], whose solution is only feasible by the use of massive-parallel processing architectures, such as graphics processing units (GPUs) and field-programmable gate arrays (FPGAs), due to the performance they can achieve and the accuracy of the results obtained, even when fixed-point operations are used [2], [8], [9]. However, hyperspectral imaging applications are computationally intensive, included artificial intelligence (AI) models, since they require a high number of operations/s [10]. Unfortunately, most studies introduce high-performance devices to deploy FPGA or GPU-based solutions [11], [12], [13] that is generally not acceptable in mobile-embedded systems,

Manuscript received 6 January 2023; revised 27 March 2023, 20 April 2023, 30 May 2023, and 15 July 2023; accepted 18 July 2023. Date of publication 24 July 2023; date of current version 3 August 2023. This work was supported in part by H2020 European Union Program under Grant 857159 (SHAPES Project) and in part by the Ministry of Economy and Competitiveness of the Spanish Government under TALENT Project PID2020-116417RB-C4 (subprojects 1 and 4) and MIRATAR Project TED2021-132149B-C41. *(Corresponding author: Julián Caba.)*

Julián Caba, Jesús Barba, Fernando Rincón, and Juan Carlos López are with the School of Computer Science, University of Castilla-La Mancha, 13071 Ciudad Real, Spain (e-mail: julian.caba@uclm.es; jesus.barba@uclm.es; Fernando.Rincon@uclm.es; juancarlos.lopez@uclm.es).

Dirk Stroobandt is with the Ghent University, 9000 Ghent, Belgium (e-mail: dirk.stroobandt@ugent.be).

María Díaz and Sebastián López are with the Institute of Applied Microelectronics (IUMA), 35017 Las Palmas de Gran Canaria, Spain (e-mail: mdmartin@iuma.ulpgc.es; seblopez@iuma.ulpgc.es).

Digital Object Identifier 10.1109/JSTARS.2023.3298484

such as UAVs, owing to the power consumption constraints due to the difficulty of heat dissipation and the low-power budget.

In this sense, the use of AI models with a particular focus on deep neural networks allows the deployment of value-added applications, which utilize a tiny fraction of the downlink bandwidth that would be, otherwise, required [14]. Existing FPGA-based AI accelerators mostly tend to increase array scale to improve throughput performance by using large FPGA devices [13], [15], but few works optimize the hardware resource utilization. The widespread adoption of AI opens up the building of commercial off-the-shelf (COTS) hardware accelerators for these algorithms, such as myriad X visual processing unit [16] and coral tensor processing unit [17], which feature high energy efficiency and remarkable performance, cost, and mass tradeoffs [18]. Furthermore, the open-source community facilitates the speed up of the deployment of the model, reducing the development time and costs with an acceptable level of reliability. Although COTS hardware accelerators are a technology that has burst onto the embedded systems, concretely in edge computing domain, the aforementioned tradeoffs obtained by these devices are still slightly worse than ones provided by an efficient massive-parallel processing architecture based on FPGA technology, as COTS hardware accelerators are general purpose devices rather than domain specific [19].

Since the bandwidth is limited, as well as the in-circuit memory and dedicated hardware resources, such as digital signal processing (DSP) units, available on devices to implement a cost- and energy-optimized solution, the way to address this challenge is to use on-board hyperspectral image compression techniques. Although there are a large variety of compression algorithms in the literature, lossless compression algorithms are preferred because they preserve most hyperspectral information [2]. Lossless techniques produce undistorted data after the decompression process, but the compression ratio is not large enough; however, near-lossless methods allow obtaining larger values of compression ratio, introducing distortions that can be controlled in accordance with the compression ratio, but it can still be too small. Nevertheless, the latest-generation sensors increase the data rate, requiring a higher compression ratio and real-time compression to avoid the accumulation of uncompressed data and, therefore, efficient transmission [20]. Although most state-of-the-art lossless compressors achieve a quite rate–distortion performance, they provide very moderate compression ratios roughly 2∼3:1 [21], which are not enough to process the amount of data produced by the newest-generation sensors. Therefore, limited communication bandwidths and increasing data volumes force to move from (near-)lossless compression techniques to lossy compression techniques, where a major research effort has been carried out in recent years [2], [22].

In mobile-embedded systems' scenarios, parallel processing devices, such as cost-optimized FPGAs, are suitable to implement hyperspectral image compression algorithms because of the balance between the degree of parallelism and the cost energy plus the cost savings of an FPGA-based solution with few resources. Unfortunately, the limited computational resources of these devices pose a new challenge when a solution is based on such technologies, so low-complexity compression schemes stand as the most practical solution for such restricted scenarios [23], [24]. Nevertheless, most state-of-the-art lossy compressors are based on the existing two-dimensional images or video compression algorithms, which are considered high computational burden, intensive memory requirements, and nonparallel nature [25]. This fact causes its use to be limited in resource-constrained environments, such as on-board compression [26]. In this context, the lossy compression algorithm for hyperspectral image systems (HyperLCA) [27] has been developed as a hardware-friendly lossy compressor for hyperspectral images, which provides good compression ratios with a reasonable computational burden, because the compression process is based on the transform operations. Briefly, this process maps the spatial domain of an image into its transformation domain to obtain the coefficients with lager amplitude, i.e., the features more representative of an image, which are then encoded. In addition, this algorithm has been designed to meet the constraints imposed by push broom/whisk broom scanners, considering each block independently, which results in less use of hardware resources. Its suitability for real-time performance applications has been previously analyzed in [28].

Nevertheless, HyperLCA and state-of-the-art lossy compressors behave linearly in compression ratio and quality performance terms, i.e., the hyperspectral data are compressed in accordance with a criterion that is defined at the beginning of the process and it does not change until it is completed. Such a criterion defines the compressed image quality as well as the computational resources used in the compression process. Concretely, signal-to-noise ratio (SNR), root-mean-square error (RMSE), and maximum absolute difference (MAD) are conventional metrics applied to lossy image compression to describe the efficiency of compression and data quality for further use [29].

In this article, the HyperLCA algorithm has been extended in order to include adaptive distortion feature without adding an overhead from the original implementation, making it a smart compressor by selecting the relevant blocks close to a predefined signature pattern. Therefore, the algorithm has been adapted to run more flexibly with different compression ratios without modifying the set of core operations performed in its original version. Furthermore, an analysis of the FPGA-based HyperLCA with adaptive distortion feature has been carried out by setting different rules of the distortion applied on the same scenario, resulting in multiplicity of compression ratios within the same image and wide range of quality compression performance. It means multiplicity rate–distortion relations take place in the compressed image, which contains high- and low-rate distortions, so not only the most different hyperspectral pixels are perfectly preserved; that is, these pixels are within the extracted information along with extra relevant pixels of interesting blocks, which results in negligible losses since some compressed blocks have less distortion. This fact benefits many hyperspectral imaging applications in which the spectral resolution is decisive; the blocks containing a large number of similarities can be compressed with a different criterion than blocks with lower similarities to the pattern. The relevant blocks are selected at runtime by comparing pixel-by-pixel with a signature pattern looking for similarities in each of the spectrum bands. Finally, the architecture has been compared with the previous version implemented in [8] and other state-of-the-art compressors in terms of throughput and hardware resource utilization. Against this backdrop, the major motivation of this work is to demonstrate the aforementioned claims about the HyperLCA with quality control, and therefore, the aim is to contribute to the scientific community with an intelligent lossy compressor for hyperspectral imaging, which enriches those blocks of greatest interest in an unexpected/unknown scenario, by using a cost-optimized and energy-efficient solution based on FPGA technology.

The rest of this article is organized as follows. Section II explains in detail the proposed quality control version of the HyperLCA compressor and highlights the differences from its original version. Section III includes a comprehensive description of the FPGA-based architecture for the execution of the compressor. The results of the proposed hyperspectral compressor implemented on a ZC7Z020 FPGA device have been evaluated in Section IV. Section V discusses about the performance and hardware resources utilization of the proposed architecture. Finally, Section VI concludes this article.

II. HyperLCA Algorithm

The HyperLCA algorithm is a lossy transform-based compressor for hyperspectral imaging, whose original version has been modified into a hardware-friendly version. Thus, the algorithm has been redesigned to achieve high compression rate–distortion ratios, as well as the computational burden has been decreased due to the high level of parallelism implemented for applications based on push broom/whisk broom sensors [9]. All of the changes made allow for the use of parallel processing architectures, such as FPGAs or GPUs, on which the algorithm has been implemented, [8] and [28], respectively. Moreover, the HyperLCA algorithm has been specially designed to work inside specific numeric ranges and use integer arithmetic, specifying the precision needed for the operations to be suitable in parallel computing architectures as previously discussed in [8], [9], and [28].

The transform-based proposed solution can independently process blocks of the image regardless of the spatial alignment of them, which facilitates the parallelization of the compression process. It means that a hyperspectral pixel can be processed independently of the subsequent pixel. On this basis, this article proposes to analyze each pixel at runtime to determine the rate of distortion that should be applied to the block being processed, rather than setting a desired minimum compression ratio for all blocks captured by the sensor, as previous versions of the algorithm performed. Fig. 1 shows an overview of the new version of the algorithm that represents the three main computing stages involved in the HyperLCA compressor with adaptive distortion feature, composed of a preprocessing or block analysis stage, a spectral transform stage, and an entropy coding stage.

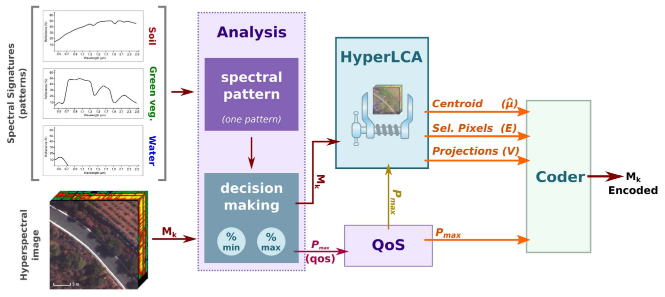

Fig. 1. Overview of the HyperLCA compressor with adaptive distortion.

A. Stage 1: Preprocessing or Block Analysis

First, the hyperspectral image is broken into blocks composed of BS consecutive horizontal pixels ($\mathbf{M}_k$), also known as lines or blocks, that will be processed. Then, the $p_{\max}$ most representative pixels within a block ($\mathbf{M}_k$) are calculated; it is feasible because the HyperLCA compressor follows the unmixing-like strategy. The previous implementations of the HyperLCA algorithm initialize the $p_{\max}$ parameter from three input parameters: CR (minimum desired compression ratio), N_{bits} (number of bits), and BS (block size) to determine the number of transformations performed on the hyperspectral block. However, in this work, the $p_{\max}$ is not fixed at design time, and it is recalculated for each block regarding to the number of similar pixels with a hyperspectral pattern signature. The balance among SNR, RMSE, and MAD quality metrics has been previously evaluated in earlier publications by combining different configurations of the input parameters (CR, N_{bits}, and BS) [28]. In particular, one of the best configurations is set the N_{bits} and BS parameters to 12 and 1024, respectively. These values are kept to calculate the $p_{\max}$ at runtime, while the CR depends on the number of hyperspectral pixels close to the pattern signature.

Therefore, the HyperLCA compressor determines the number of transformations performed on the block being processing, $\mathbf{M}_k$, as shown in (1), where *DR* refers to the number of bits per pixel per band; nb represents the number of bands; BS is the number of consecutive horizontal pixels in a single block; CR refers to the desired compression ratio, defined as the relation between the number of bits in the original image and the ones of the compressed data; and N_{bits} is the number of bits that determines the precision and dynamic range to be used for representing the values of the compressed data. Thus, the number of transformations performed $p_{\max}$ directly determines the maximum compression ratio to be reached with the selected configuration in which higher $p_{\max}$ values result in better reconstructed images, but lower compression ratios

$$p_{\max} \leq \frac{DR \cdot nb \cdot (BS - \text{CR})}{\text{CR} \cdot (DR \cdot nb + N_{\text{bits}} \cdot BS)}. \tag{1}$$

The compression ratio (CR) used in the calculation of $p_{\max}$ for a block must be determined by analyzing each of the pixels of that block ($\mathbf{M}_k$), comparing them one-by-one with a reference hyperspectral signature (reference pattern). This process can be done by using the Euclidean distance with the reference

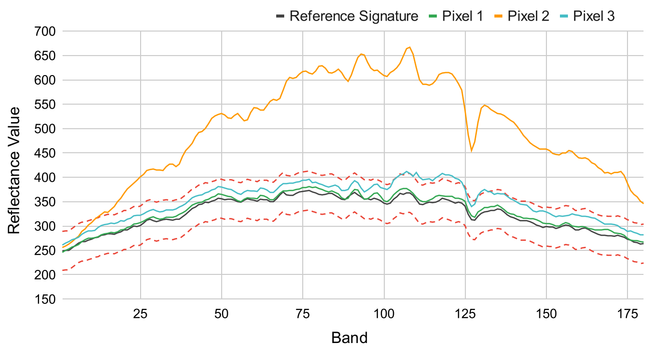

Fig. 2. Overview of pixel selection process by limits.

signature and the hyperspectral pixels within a block to determine if the block, which is being processed, contains a high percentage of pixels close to the reference or, on the contrary, contains few matches. However, Euclidean distance calculation is costly in reconfigurable devices and makes it necessary to look for alternatives. In this sense, this work proposes a hardware-friendly solution by the use of band-limit values, where a *delta* value is defined to set the upper and lower limits that draw the same plot that of the reference signature. The results of this method are similar to the ones obtained by the Euclidean distance method.

Fig. 2 shows graphically the band-limit adopted approach, where three hyperspectral pixels and a reference signature (black line) are plotted. In addition, the red dotted lines define the upper and lower limits whose trend is identical to that drawn by the reference signature. Therefore, hyperspectral signatures within both limits are considered close to the reference (green line of Fig. 2), as well as those pixels whose a few number of reflectance values are outside the limits (blue line of Fig. 2); the number of values outside the limits is configurable. This fact means that the hyperspectral signatures whose reflectance is very similar to the reference, except in a small number of bands, are also considered to be close to the reference signature. Meanwhile, the hyperspectral pixels whose most reflectance values are outside the limits are not considered (orange line of Fig. 2).

Once all hyperspectral pixels within a block are defined as in or out the limits, the CR is set by analyzing the number of pixels inside the boundaries. Fig. 3 shows three examples with different compression ratios over the same scenario. First, Fig. 3(a) shows an original RGB figure of a vineyard scenario, while Fig. 3(e) shows a pattern in which white and black colors represent the green vegetable signatures and other materials, respectively. Unfortunately, the sensor used in this work captures the hyperspectral information line-by-line; it means the whole hyperspectral information of a scenario is not obtained immediately, so the operations must be performed over a line ($\mathbf{M}_k$) with BS hyperspectral pixels. Thus, when a low distortion is required but only in the blocks which contain useful information for the final application, the number of hyperspectral pixels close to the reference must be small. In this sense, Fig. 3(b)–(d) highlight in green the blocks whose CR is low, i.e., these lines contain useful information for the final application, while the blocks whose hyperspectral information is not too relevant are highlighted in black. Meanwhile, Fig. 3(f)–(h) highlight the loss, hit, and

Algorithm 1: HyperLCA Transform.

```
 Inputs:
 $\mathbf{M}_k = [\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_{BS}]$, $p_{\text{max}}$
 Outputs:
 $\hat{\boldsymbol{\mu}}$; $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_{p_{\text{max}}}]$; $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{p_{\text{max}}}]$
 Algorithm:
 1: Average pixel: $\hat{\boldsymbol{\mu}}$;
 2: Centralized image: $\mathbf{C} = \mathbf{M}_k - \hat{\boldsymbol{\mu}} = [\mathbf{c}_1, \mathbf{c}_2, \ldots \mathbf{c}_{BS}]$;
 3: for $n = 1$ to $p_{\text{max}}$ do
 4:    for $j = 1$ to $BS$ do
 5:       Brightness calculation: $\mathbf{b}_j = \mathbf{c}_j' \cdot \mathbf{c}_j$;
 6:    end for
 7:    Maximum brightness: $j_{\text{max}} = \text{argmax}(\mathbf{b}_j)$;
 8:    Extracted pixels: $\mathbf{e}_n = \mathbf{r}_{j_{\text{max}}}$;
 9:    $\mathbf{q}_n = \mathbf{c}_{j_{\text{max}}}$;
10:    $\mathbf{u}_n = \mathbf{q}_n / b_{j_{\text{max}}}$;
11:    Projection vector: $\mathbf{v}_n = \mathbf{u}_n' \cdot \mathbf{C}$;
12:    Information subtraction: $\mathbf{C} = \mathbf{C} - \mathbf{q}_n \cdot \mathbf{v}_n$;
13: end for
```

excess of hyperspectral information with respect to the ideal information extraction [see Fig. 3(e)] in white, green, and red, respectively. The process to select relevant blocks in the different scenarios fulfills the following criteria: at least ten pixels close to the reference for the scenario represented in Fig. 3(b), at least 300 pixels close in the case of Fig. 3(c), and at least 800 pixels in the last case [see Fig. 3(d)].

B. Stage 2: Spectral Transform

The HyperLCA compressor is among the transform-based algorithms that employ a modified version of the well-known Gram–Schmidt orthogonalization method, which is widely used in lossy compression because of its moderate complexity in which no band reordering is required. The basic idea of this class of algorithms is to map the spatial domain of a hyperspectral image into its transformation domain [2]. In this regard, the *Spectral Transform* stage selects the most different pixels of a hyperspectral image using orthogonal projection techniques. Therefore, the selected pixels are used for projecting the image in order to remove redundancies and, thus, obtain a spectral decorrelated and compressed image.

The *Spectral Transform* applied by the HyperLCA compressor is described in detail in Algorithm 1. The inputs of this stage are the hyperspectral block to compress ($\mathbf{M}_k$), which is the same as the one being analyzed by the *preprocessing* stage, and the number of most different pixels to extract (p_{max}), which is the output of the *preprocessing* stage, i.e., it is responsible for determining the number of transformations or orthogonal projections to perform on the hyperspectral block. While the outputs of this stage are the average pixel ($\hat{\boldsymbol{\mu}}$) of the input hyperspectral block ($\mathbf{M}_k$), the set of indexes related to the p_{max}, most different hyperspectral pixels ($\mathbf{E}$), and their corresponding projection vectors ($\mathbf{V}$).

The first step of the algorithm is to centralize the hyperspectral block ($\mathbf{M}_k$) subtracting the average pixel ($\hat{\boldsymbol{\mu}}$), which is obtained by adding all the pixels of a band of the hyperspectral block

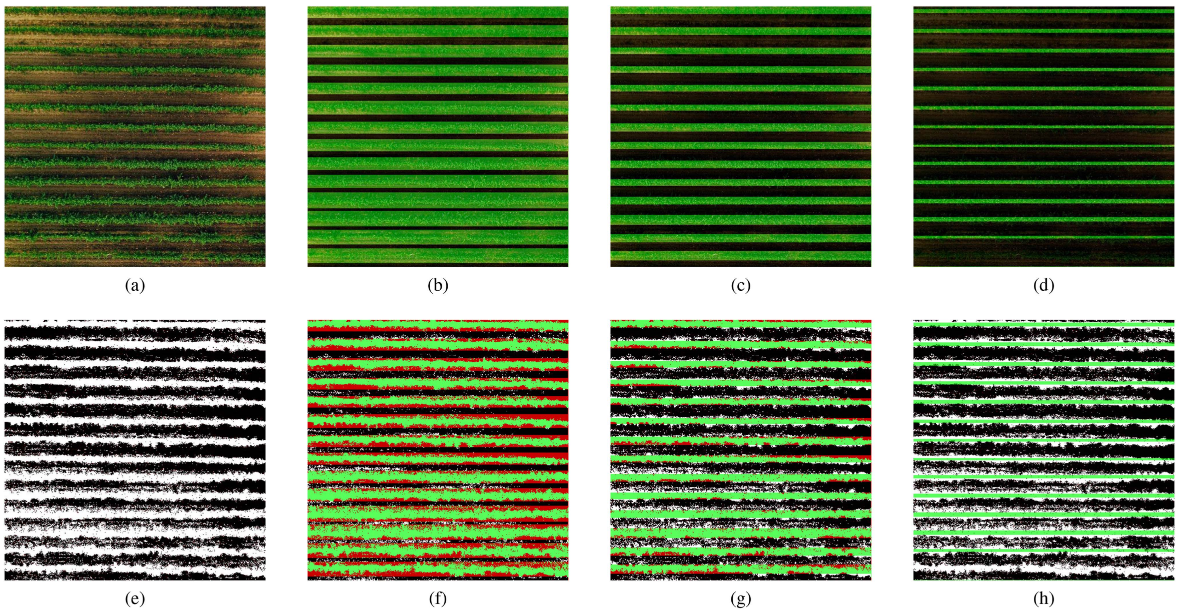

Fig. 3. Pixel selection in a vineyard scenario (selected blocks in green color). (a) Original vineyard. (b) Line selection with low-compression ratio. (c) Line selection with midcompression ratio. (d) Line selection with high-compression ratio. (e) Pattern of pixel selection. (f) Differences between selected pixels in (b) and pattern. (g) Differences between selected pixels in (c) and pattern. (h) Differences between selected pixels in (d) and pattern.

and dividing by the number of pixels (BS); this operation is performed for all the bands of the block, resulting in a vector with nb (number of bands) elements (line 1 of Algorithm 1). Then, it is used to get the centralized version of the hyperspectral block ($\mathbf{C}$) in line 2. In the second step, the $p_{\max}$ most characteristic pixels are sequentially extracted from lines 3 to 13. For this purpose, the brightness of each hyperspectral pixel within the block is calculated following a vector normalization strategy, which is defined as the product of each band within the hyperspectral pixel with itself (lines 4–6 of Algorithm 1). Subsequently, the highest brightness ($\mathbf{j}_{\max}$) in each iteration determines the pixels to extract ($\mathbf{e}_n$) from the original hyperspectral block ($\mathbf{M}_k$), so the algorithm searches for the maximum value of the previously calculated brightness and then extracts the original pixel whose index matches this maximum brightness; such operations are performed in lines 7 and 8 of Algorithm 1, respectively. Afterward, the orthogonal vectors $\mathbf{q}_n$ and $\mathbf{u}_n$ are obtained (lines 9 and 10); $\mathbf{q}_n$ is the hyperspectral pixel of the block ($\mathbf{M}_k$) whose brightness value is the highest, while $\mathbf{u}_n$ is obtained by dividing each value of $\mathbf{q}_n$ by the brightness value. The projection of each block pixel over the direction spanned by the selected pixel is estimated with $\mathbf{u}_n$ orthogonal vector (line 11 of Algorithm 1). Finally, the extracted information is subtracted from the centralized block $\mathbf{C}$ in line 12, resulting in a new hyperspectral block, also called $\mathbf{C}$, that is the new input of the loop block.

Accordingly, $\mathbf{C}$ retains the spectral information that is not represented by the extracted pixels ($\mathbf{E}$) and, thus, it also represents the information that will not be recovered in the decompression process. Therefore, the highest $p_{\max}$ values reduce the information that cannot be recovered. In this sense, the adaptive distortion feature mitigates the lost of information in the relevant blocks for the final application, so higher $p_{\max}$ values are applied to those blocks.

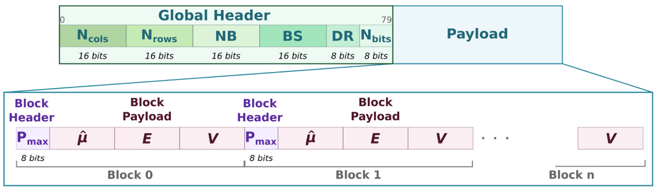

Fig. 4. Overview of the bitstream structure generated.

C. Stage 3: Entropy Coding

The last stage of the HyperLCA compressor performs the entropy coding of the vectors received from the *Spectral Transform* stage, where the use of Golomb–Rice algorithm [30] makes possible to consider each vector independently from the rest, i.e., the outputs of *Spectral Transform* stage can be consumed as they are received; thus, both stages are carried out in parallel. To do so, the compression parameter (N) is calculated as the average value of the vector being processed. Afterward, the elements of the vector are divided by N as well as the quotient (q) and the remainder (r) of such operation are also obtained. Second, the lowest power of 2 higher than N is calculated as $b = \log_2(N) + 1$. The quotient (q) is codified using unary code, while the remainder (r) is coded as plain binary using $b - 1$ bits for r values smaller than $2^b - N$; otherwise, it is coded as $r + 2^b - N$ using b bits.

D. Bitstream Generation

Finally, the *Data Packaging* stage is performed to generate a unique bitstream that contains the outputs of the aforementioned compression stages. Fig. 4 graphically shows the structure generated for the compressed bitstream, which is divided into two hierarchical blocks.

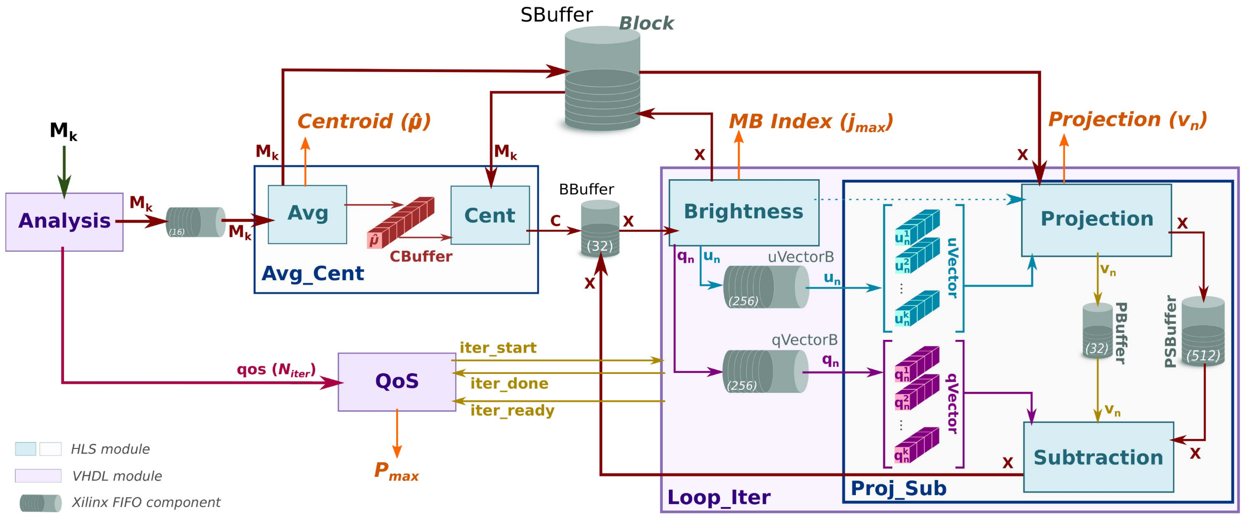

Fig. 5. Overview of the hardware implementation related to the HyperLCA transform with quality control.

First, a *Global Header* takes place at the beginning of the stream to define the global information about the hyperspectral image, including the parameters used in the compression process. It contains the spatial (number of columns and rows) and spectral (number of bands) information of the processed image, and it also denotes the block size used (BS), the number of bits per pixel per band (DR), and the number of bits used for representing the values of the compressed data (N_{bits}).

The payload is placed after the *Global Header*, which is broken into blocks that contains two fields: *Block Header* and *Block Payload*. Owing to the fact that the blocks of a hyperspectral image are compressed with a variety number of transformations, it is mandatory to have an 8-bit header to determine the $p_{\max}$ applied in each block. After the *Block Header*, there is the *Block Payload,* which is composed of a single centroid regardless of the distortion applied on the block and then the vectors $\mathbf{E}$ (most different pixels) and $\mathbf{V}$ (projections), whose number is denoted by the aforementioned 8-bit header.

III. FPGA Implementation of HyperLCA Algorithm

The HyperLCA compressor with adaptive distortion feature has been implemented on reconfigurable hardware using high-level synthesis (HLS) to define each operation involved in the HyperLCA algorithm as a set of specialized hardware accelerators. HLS reduces the development lifecycle due to the use of high-level languages, such as C or C++, to describe the functionality and include features that allow to optimize and improve the functionality described by nonhardware engineers [31].

In this work, specialized hardware accelerators have been reused from [8] to build the new architecture of the HyperLCA transform, adding the dynamic behavior required to include multiple compression ratios. Fig. 5 shows an overview of the hardware architecture implemented for *preprocessing* and *spectral transformation* stages, where the blue boxes correspond to the modules described in HLS, and are, therefore, inherited from the previous development. These modules are connected through memory buffers and custom logic that orchestrates them to obtain the desired behavior. For the sake of clarity, the matching of the blue boxes with the operations performed in Algorithm 1 is as follows: *Avg_Cent* corresponds to lines 1 and 2, while *loop_iter* is the main loop body that comprises from lines 3 to 13, where *brightness* is calculated in the inner loop and line 7 of Algorithm 1; orthogonal vectors ($\mathbf{q}$ and $\mathbf{u}$) are also obtained once the brightest pixel is selected. Meanwhile, *projection* and *subtraction* match with lines 11 and 12 of Algorithm 1, respectively. In pursuit of improving the performance, the architecture introduces an enhancement through the partitioning of the orthogonal vectors, $\mathbf{q}$ and $\mathbf{u}$, using VHDL instead of applying a pragma directive on the HLS description, just before they are used by the *projection* and *subtraction* modules.

Up to this point, the described architecture is responsible to carry out the transformation process on the hyperspectral blocks ($\mathbf{M}_k$). However, the transform-based operations carried out, i.e., the number of executions of the *loop_iter* module is now calculated at runtime. The number of iterations is related to the compression ratio applied to a block, which also means the amount of hyperspectral data is represented for each block. Thus, the *QoS* module manages this process as well as notifies the $p_{\max}$ value used to encode the current hyperspectral block ($\mathbf{M}_k$). In contrast, the *Analysis* module is in charge of calculating the number of iterations to be performed according to the similarity between the pixels that compose a hyperspectral block and the pattern signature.

Therefore, the *Analysis* module corresponds to *Stage 1* (preprocessing) and it is the first operation to be executed before performing any transformation. In fact, the block analysis process and the first operation of the *spectral transform* stage (i.e., the calculation of the average pixel) work in parallel. To do so, the *Analysis* module makes a copy of the hyperspectral block ($\mathbf{M}_k$) in batch mode as soon as it receives the block, so the *Avg* module can begin to process it. The parallelism of both operations allows to obtain the number of transformations to be applied

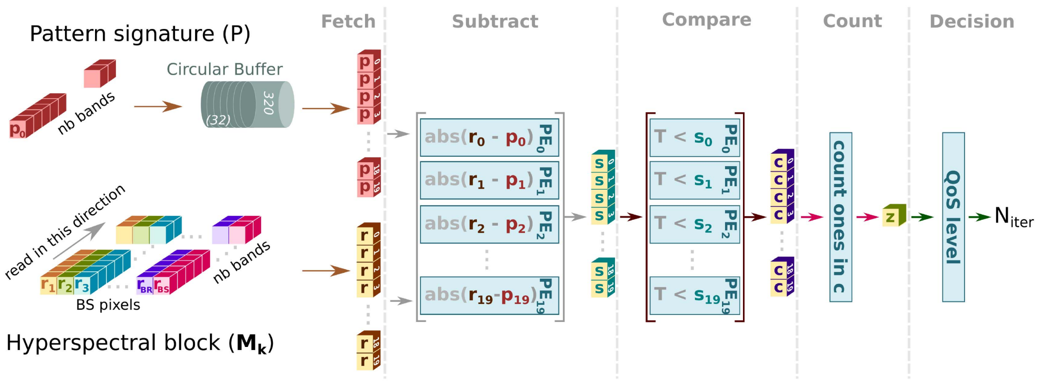

Fig. 6. Overview of pipeline-based architecture to calculate the number of transformations (Number of PEs = 20).

to the current hyperspectral block with a negligible overhead and it does not block the HyperLCA operations. Focusing on the hardware implementation of the *Analysis* module, the operations performed by this module have been segmented into a four-stage pipeline and an additional decision stage that determines the number of iterations (N_{iter}) to apply in the *spectral transform* stage. Fig. 6 shows an overview of the pipeline-based architecture, which has been implemented in VHDL and whose stages perform the following tasks.

Fetch: The data of the pattern signature ($\mathbf{P}$) and hyperspectral block ($\mathbf{M}_k$) are read in batch mode and stored in p_i and r_i registers, respectively (see Fig. 6). Thus, this stage reads as many bands as can be processed in parallel by the operators of the *spectral transform* stage, e.g., 20 processing elements (PEs) are required to process 20 spectral bands in parallel. In this sense, the word widths of the internal memories/registers of the architectures, as shown in Figs. 5 and 6, must be configured to work with the corresponding number of bands. On top of that, the pattern signature, which is previously stored in a ROM, is moved to a circular buffer before being retrieved; this process is performed before the *fetch* stage and it is done to save time and improve the performance because the pattern must be used the same number of times as the number of hyperspectral pixels contained in the captured image, i.e., the pattern signature is read BS times in a hyperspectral block ($\mathbf{M}_k$) and repeated by the lines that contain the hyperspectral image.

Subtract: Once the hyperspectral data are ready, the *subtract* stage subtracts from the current hyperspectral block ($\mathbf{M}_k$) the pattern signature by bands so that the first band of the hyperspectral pixel belonging to the processed block is subtracted by the first band of the stored pattern, then the second band of both, and so on. These operations are also performed in parallel. Finally, the absolute value of the operation is also calculated. The results of this stage are stored in s_i registers, which are represented as yellow and green boxes in Fig. 6.

Compare: The results obtained from each PE in the *subtract* stage are compared with a constant value T, which is the predefined *delta* value. Thus, if the constant T is smaller than the result of the absolute subtraction obtained in the previous stage, it means that the corresponding band is within the limits. On the other hand, if the comparison results that T is greater, it implies that the value of the band is outside the limits. The output of this stage is a zero or one when the band is outside or within the limits, respectively. In a similar way to the previous stage, the results of the comparisons are also stored in c_i registers (yellow and purple boxes in Fig. 6).

Count: In this stage, the registers that contain a one, i.e., the band is within the limits, are counted and added to a partial sum, which is stored in z register (see Fig. 6). Thus, this register is updated until the hyperspectral pixel is compared with the pattern signature, then it is set to zero to compare the next pixel of the hyperspectral block.

Finally, the *decision* stage is performed after the whole hyperspectral pixel of a block is compared with the pattern signature. It determines whether a pixel is close to the pattern from the sum value stored in the z register and calculated by the *count* stage. In this sense, it is not necessary that all band values are within the limits, the solution allows several bands to be out of them (see blue line in Fig. 2). On top of that, the *decision* stage sets the number of transform operations carried out by the *spectral transform* stage. It is calculated by counting the number of pixels close to the pattern signature (T_c) and, then, compare such sum with the predefined user rules, which figures out the provided quality of service, i.e., the degree of distortion. To do that, the user must provide two constants R_{min} and R_{max} to apply the following cases, where QoS_{max}, QoS_{min}, and QoS_{med} are the number of iterations that are also predefined; a high number means greater spectral information extracted

$$\begin{cases} N_{\text{iter}} = \text{QoS}_{\text{max}}, & \text{if } T_c \geq R_{\text{max}} \\ N_{\text{iter}} = \text{QoS}_{\text{min}}, & \text{if } T_c \leq R_{\text{min}} \\ N_{\text{iter}} = \text{QoS}_{\text{med}}, & \text{otherwise.} \end{cases}$$

IV. Results

The lossy compressor with adaptive distortion feature is evaluated in this section by analyzing the FPGA-based architecture implemented on a ZC7Z020-CLG484 and the performance achieved by it, comparing the throughput and power consumption when several PEs are working in parallel. This section also evaluates the accuracy of the limit method versus the use of Euclidean distance, which involves complex operations, such as square root operation, that demand high computational resources when it is developed in reconfigurable technology.

A. Hyperspectral Dataset Collected

The performance of the proposed FPGA-based architecture has been evaluated by a set of hyperspectral images, which were sensed by a custom aerial platform over different farming areas on the island of Gran Canaria (Spain). The dataset contains four hyperspectral images collected over two different vineyard areas, whose exact coordinates are $27°59'35.6''$N $15°36'25.6''$W and $27°59'15.2''$N $15°35'51.9''$W. For analysis purposes, the dataset used in the previous work is kept [8].

The acquisition platform mounts a *Specim FX10* push broom hyperspectral camera on a DJI Matrice 600 drone [32]. The image sensor captures 1024 spatial pixels per track and up to 224 spectral bands in the range between 400 and 1000 nm. Nevertheless, the experiments performed only work with 180 spectral bands; the first 10 spectral bands are discarded as well as the last 34 bands. This fact is done because the response in the boundaries of the electromagnetic spectrum is low.

B. Evaluation of the Near Pixel Selection Method

The goodness of the pixel selection method through upper and lower limits proposed in this work has been evaluated and compared with the Euclidean distance. For doing so, the analysis has been carried out using the set theory operations to determine the similarity of the results obtained from both solutions. Concretely, six different metrics have been applied to analyze the set of pixels selected by the method used in this work. It is worth mentioning that the result obtained, regardless of the method used, is a set of pixels that are similar to the reference pattern, and hence, the analysis must be driven by the similarities between result sets.

In this context, the universal set $\mathbf{U}$ is composed of the index of pixels within a hyperspectral block ($\mathbf{M}_k$). Thus, $\mathbf{U}$ is defined as $\mathbf{U} = \{0, \ldots, 1023\}$ or $\mathbf{U} = \{x \mid x \in \mathbb{N} \wedge x \leq 1023\}$. The set of pixels selected by the Euclidean distance algorithm is denoted as $\mathbf{E}$ ($\mathbf{E} \subseteq \mathbf{U}$), whereas the ones extracted by the limit method are represented as $\mathbf{L}$ ($\mathbf{L} \subseteq \mathbf{U}$). The following lines describe the six metrics used to determine the degree of similarity of the two approaches.

True Positives: This metric analyzes the number of common pixels selected by the two algorithms, i.e., the intersection operation is applied to $\mathbf{E}$ and $\mathbf{L}$ sets, which are obtained by the Euclidean distance and the limit method, respectively [see (2)]. Equation (3) calculates the percentage of hits in $\mathbf{L}$ relative to $\mathbf{E}$; a high percentage means that most pixels selected by the Euclidean distance are also chosen by the limit method, but it does not imply that they are very similar; the $\mathbf{L}$ set may contain many more pixels than $\mathbf{E}$ does not

$$E \cap L = \{x \mid x \in E \wedge x \in L\} \tag{2}$$

$$n(E \cap L)/n(E). \tag{3}$$

False Negatives: In this case, it represents the pixels within $\mathbf{E}$ but not in $\mathbf{L}$, or in other words, it denotes the pixels that the limit algorithm should have included in its solution set but are not actually included. The set of false negatives is extracted by the difference operation as (4) shows, where the common pixels of $\mathbf{E}$ and $\mathbf{L}$ are extracted from $\mathbf{E}$. This metric and the true positive metric provide the degree of similarity between sets; a low percentage of false negatives and a high percentage of true positive means that two sets are very similar. Equation (5) calculates the percentage of pixels not considered by the limit method

$$E - L = \{x \mid x \in E \wedge x \notin L\} \tag{4}$$

$$n(E - L)/n(E). \tag{5}$$

False Positives: This metric measures the degree of extra pixels considered by the limit method. It is also calculated by the difference operation but now the common pixels of $\mathbf{E}$ and $\mathbf{L}$ are extracted from $\mathbf{L}$ instead of $\mathbf{E}$ [see (6)]. A small percentage value of this metric implies that the solution is tight, whenever the true positives metric has a high value and the false negatives metric has a low value. The percentage of pixels included by the limit method is calculated in (7)

$$L - E = \{x \mid x \notin E \wedge x \in L\} \tag{6}$$

$$n(L - E)/n(L). \tag{7}$$

True Negatives: In this case, the metric denotes the pixels not included in the solutions of both methods, i.e., the pixels that are neither in $\mathbf{E}$ nor $\mathbf{L}$. It is calculated by the union of $\mathbf{E}$ and $\mathbf{L}$ sets and then the result is subtracted (difference operation) from the universal set ($\mathbf{U}$). Equation (8) shows the operations performed to obtain the set of pixel considered as true negatives, while (9) shows the formula to calculate the success rate of this metric, where E' is the true negatives of the result obtained by the Euclidean distance algorithm

$$U - (E \cup L) = \{x \mid x \in U \wedge (x \notin E \vee x \notin L)\} \tag{8}$$

$$n(U - (E \cup L))/n(E'). \tag{9}$$

$\mathbf{E}$ *is a subset of* $\mathbf{L}$*:* This metric determines whether the whole set obtained from the Euclidean distance algorithm is a subset of the set obtained by the limit method. It means that the solution contains all pixels that the reference algorithm.

$\mathbf{L}$ *is a subset of* $\mathbf{E}$*:* In this case, it denotes the lack of pixels in the solution because the set obtained by the Euclidean distance algorithm is greater than the one obtained by the limit method. This metric and the above one are interesting to attend when a large number of pixels are analyzed; with two sets of pixels, this information can be extracted from the first four metrics.

Table I lists the percentage of similarity between the Euclidean distance and limit methods to select the set of pixels close to the pattern signature. To do so, the set obtained by the Euclidean method must be satisfied that none of the members of such set exceeds the value of 200 with respect to the distance to the reference. Meanwhile, the limit method defines the upper and lower boundaries according to the spectral values of the pattern signature with a unique variable (*limits* column of Table I); in addition, it also defines the maximum number of spectral bands whose values are outside the boundaries (*errors* column of

TABLE I
COMPARISON OF RESULTS OBTAINED BETWEEN THE EUCLIDEAN DISTANCE AND LIMIT METHODS

Limits	Errors	True Positives ($\mathbf{E} \cap \mathbf{L}$)	False Negatives ($\mathbf{E} - \mathbf{L}$)	False Positives ($\mathbf{L} - \mathbf{E}$)	True Negatives ($\mathbf{U} - (\mathbf{E} \cup \mathbf{L})$)	$\mathbf{E} \subseteq \mathbf{L}$	$\mathbf{L} \subseteq \mathbf{E}$
±15	20	49.093%	50.906%	0.0000%	100.00%	0.6835%	100.00%
	25	54.467%	45.532%	0.0000%	100.00%	0.7812%	100.00%
	30	60.003%	39.996%	0.0077%	99.999%	1.0745%	99.804%
±20	20	82.158%	17.841%	0.2145%	99.992%	6.3476%	93.261%
	25	89.024%	10.975%	0.9618%	99.963%	14.160%	73.339%
	30	94.686%	5.3133%	2.8799%	99.879%	28.711%	41.601%
±23	20	96.843%	3.1564%	7.6101%	99.658%	42.187%	22.949%
	25	99.256%	0.7437%	12.146%	99.413%	76.172%	8.0078%
	30	99.958%	0.0418%	17.789%	99.074%	98.339%	2.6369%
±24	20	98.703%	1.2969%	12.198%	99.413%	65.332%	11.230%
	25	99.823%	0.1766%	17.296%	99.106%	93.652%	3.0273%
	30	99.995%	0.0046%	22.758%	98.739%	99.804%	1.3671%
±25	20	99.526%	0.4741%	16.796%	99.140%	85.156%	4.0039%
	25	99.944%	0.0557%	22.292%	98.773%	97.949%	1.5625%
	30	100.00%	0.0000%	27.498%	98.377%	100.00%	0.7812%
±26	20	99.842%	0.1580%	21.478%	98.831%	94.443%	2.2461%
	25	99.991%	0.0093%	26.839%	98.430%	99.609%	0.8789%
	30	100.00%	0.0000%	31.697%	98.014%	100.00%	0.6836%
±27	20	99.939%	0.0604%	25.837%	98.510%	97.754%	1.0742%
	25	100.00%	0.0000%	30.963%	98.081%	100.00%	0.6836%
	30	100.00%	0.0000%	35.542%	97.641%	100.00%	0.6836%
±30	20	100.00%	0.0000%	37.044%	97.482%	100.00%	0.4883%
	25	100.00%	0.0000%	41.147%	97.008%	100.00%	0.3906%
	30	100.00%	0.0000%	44.861%	96.519%	100.00%	0.2929%

Table I). These parameters are configurable and, for the current analysis, the selected values are listed in the two first columns of Table I.

Analyzing the results, as shown in Table I, the best configurations are those with a high percentage of true positives and a low percentage of false positives and negatives. In this sense, the configurations with 100% hits (true positives) have at the same time the highest percentage of false positives, so they are not good candidates, e.g., the last limit configuration listed in the table. On the other hand, configurations with a low percentage of false negatives and an acceptable percentage of true positives are not good candidates either, for example, the second configuration of the limits (Limits $= \pm 20$) reaches up to 94.68% in true positives, but the value of the $\mathbf{E} \subseteq \mathbf{L}$ metric is very low; 6.34%, 14.16%, and 28.71% for 20, 25, and 30 *Errors* parameter configuration, respectively. Thus, it means that most of the obtained sets do not include all values obtained by the Euclidean distance method. Focusing on the rest of the configurations, the best configuration is the one whose *Limits* and *Errors* parameters are configured with ± 25 and 30, respectively. The obtained results with the aforementioned configuration contain all pixels acquired by the Euclidean distance but extending the sets by including more pixels, as it denotes the false positives metric. In addition, the second best configuration, Limits $= \pm 24$ and Errors $= 30$, is also a very good candidate because of its balance between true positives and false positives. Therefore, the limit method is a suitable candidate to replace the Euclidean distance method in reconfigurable devices due to the simplicity of the operations performed, as well as the similarity of the obtained results.

C. Hardware Analysis

The proposed architecture has been implemented on a ZC7Z020-CLG484 FPGA device for analysis purposes and with a twofold objective; first, the architecture is kept in order to compare this new version of the algorithm with the one previously implemented in [8]; second, the FPGA architecture (Artix) is a cost-optimized device compared with other architectures of the same manufacturer, such as Kintex or UltraScale/UltraScale+. In this sense, the selected device provides a good tradeoff on three aspects: cost, power consumption, and throughput. However, to achieve a good ratio balance in performance/W, it is necessary to invest engineering efforts in the architectural part as a consequence of the limited resources of the ZynQ SoC. For the sake of clarity, the hyperspectral images have been stored in an external memory (DDR) to analyze the performance of the developed architecture. Thus, the hardware accelerator reads the spatial and spectral information of the hyperspectral images from the DDR through an AXI-Stream interface using a direct memory access.

TABLE II
HARDWARE UTILIZATION OF HYPERLCA ALGORITHM WITH ADAPTIVE DISTORTION FEATURE FOR AN XC7Z020-CLG484 SOC AFTER POSTIMPLEMENTATION PHASE

PEs	BRAM18K	DSP48E	Flip -flops	LUTs
1	211 *(75.36%)*	9 *(4.09%)*	6146 *(5.78%)*	7502 *(14.10%)*
2	208 *(74.29%)*	16 *(7.27%)*	6186 *(5.81%)*	8329 *(15.66%)*
4	216 *(77.14%)*	30 *(13.64%)*	7048 *(6.62%)*	9384 *(17.64%)*
6	223 *(79.64%)*	62 *(28.18%)*	8134 *(7.64%)*	10 886 *(20.46%)*
10	232 *(82.86%)*	102 *(46.36%)*	9684 *(9.10%)*	12 733 *(23.93%)*
12	217 *(77.50%)*	122 *(55.45%)*	10 573 *(9.94%)*	14 458 *(27.18%)*
20	218 *(77.86%)*	202 *(91.82%)*	13 834 *(13.00%)*	19 178 *(36.05%)*

Table II lists the programmable logic (PL) resources in accordance with the number of PEs instantiated in each configuration after postimplementation phase, where the PEs work in parallel to process several bands in batch mode. The hyperspectral block size (BS) is set to 1024 hyperspectral pixels, while the spatial size is 180 bands. The number of PEs that can be instantiated is 20 PEs since the number of available DSP units is 220 in the ZC7Z020-CLG484 FPGA device and such configuration requires 202 of these units. Thus, the DSPs are the limiting resource of the proposed architecture because the next possible configuration instantiates 30 PEs, so it will demand 257 DSPs,

TABLE III
MSamples/s Achieved by the Different Versions of the HyperLCA Accelerator With Adaptive Distortion Feature for an XC7Z020-CLG484 SoC

PEs	1	2	4	6	10	12	20
100 MHz	58	120	236	332	494	562	777
143 MHz	87	180	352	495	734	835	1149

TABLE IV
Power Consumption of the Different Versions of the HyperLCA Accelerator With Adaptive Distortion Feature for an XC7Z020-CLG484 SoC

	PEs	1	2	4	6	10	12	20
100MHz	**MSs/W**	25	54	106	141	201	232	305
	PL (W)	0.679	0.648	0.671	0.796	0.893	0.862	0.993
	Temperature (ºC)	50.7	50.4	50.6	52.1	53.2	52.8	57.0
143MHz	**MSs/W**	36	77	149	195	273	316	413
	PL (W)	0.809	0.759	0.807	0.985	1.131	1.082	1.225
	Temperature (ºC)	52.2	51.8	52.2	54.3	55.9	55.4	57.0

i.e., a device with more hardware resources is requested. It is worth mentioning that the number of PEs must be a divisor of the number of bands to properly apply the optimizations. Meanwhile, the rest of the hardware resources are not critical for the scalability of the architecture, although the BRAMs are between 74% and 83%; it really depends on the block size parameter (BS), whose value is set at design time.

The throughput of each configuration of the hardware accelerator is depicted in MSamples/s, which depends on the number of PEs instantiated and the clock frequency configuration. In this sense, the RTL models for the HLS modules were generated setting the target clock frequency at 100 MHz. Then, two versions of the bitstream were synthesized for two different clock configurations: 100 and 143 MHz. From previous works, the authors observed that, in some cases, the model generated by the HLS tools differs greatly due to the characteristics of the generation process, resulting in higher scheduling cycles for shorter clock periods. Nevertheless, the Vivado tool was able to handle the better 100 MHz RTL generated models and synthesize a valid bitstream for other clock frequencies, fully compliant with the platform timing constraints. Table III lists the average of the MSamples/s achieved by the hardware accelerator according to the number of PEs and the configuration of the clock frequency. It is worth mentioning that the MSamples/s metric mainly depends on two configuration parameters of the hardware accelerator: the pattern signature and the rules that determine the compression ratio applied to the current hyperspectral block. Thus, the MSamples/s values of Table III are an average of the results obtained from the four sensed images by the UAV platform by applying three rules, where the fast versions (143 MHz) achieve 33% better performance than the slow ones (100 MHz).

The hyperspectral sensor used to sense the images is based on a *Specim FX10*, which is a push broom camera that can be configured to set the size of each frame and the spatial information sensed, i.e., the block size (BS) and the number of hyperspectral bands, respectively. The BS has been configured to the maximum allowed by this camera (1024 pixels) because Diaz et al. [28] analyze the image quality by comparing the SNR, RMSE, and MAD metrics to determine the quality of the compression process with the same camera and conclude that the best results are obtained with the maximum block size. Thus, the selected hyperspectral sensor can capture 1024 pixels with 224 bands with a maximum frame rate of 327 fps (full range) [33], while the frame rate obtained to capture images of 1024 pixels with 180 bands is approximately 400 fps. Therefore, analyzing the performance results and the features of the hyperspectral camera used to sense the images, the proposed architecture is able to compress hyperspectral information at runtime with six PEs and the clock frequency configured at 143 MHz or with ten PEs with the clock frequency set at 100 MHz.

From the point of view of energy efficiency, the number of PEs is the factor that increases the power requirements, as well as the working temperature. Table IV lists the energy efficiency depicted in MSamples/s per watt (MSs/W), i.e., the power consumption obtained by the FPGA-based implementation of the architecture according to the PEs and clock frequency parameters, the power consumption of the hardware accelerator in W (only the HyperLCA+QoS core), and the working temperature in Celsius grades. The values listed in Table IV have been reported by the AMD-Xilinx for the worst-case scenario after postimplementation stage, showing the variation of the different parameters as the SoC configuration increases the number of PEs. To do the accurate analysis of the worst-case scenario, the parameters of the process have been set to a maximum value in order to obtain the power consumption and thermal values in such a scenario. This way, the power and thermal delivery solution will work with any device that will be shipped [34]. Regarding the energy efficiency, the MSs/W tradeoff has been calculated with the power consumption, including the embedded microcontroller, i.e., the processing system (PS) and PL parts of the targeted device. It does not draw a linear behavior since the number of PEs working in parallel increases the number of hardware resources, as well as the power consumption of the FPGA part; the energy budget of the static part is stable. In this sense, Table IV also presents the power utilization by the hardware implementation of the HyperLCA algorithm, which demonstrates that the architecture is highly efficient because the main energy budget is consumed by the hardcoded embedded processor (1.533 W), so the extra power of the FPGA-based implementation needed is ranging from 0.314 W and 0.416 W for the 100 MHz and 143 MHz versions, respectively. On top of that, the working temperature does not have a major impact on the 143 MHz versions of the system or on the number of PEs working in parallel.

Although Vivado's power analysis tools provide the accurate power-consumption results after postimplementation phase, there are other hardware components outside the FPGA device that are involved in the compression hyperspectral process, such as external memory (RAM), that are not considered in the

TABLE V
POWER CONSUMPTION OF THE DESIGN ON ZEDBOARD (XC7Z020-CLG484) WITH CURRENT SENSE (J21 CONNECTOR)

	PEs	1	2	4	6	10	12	20
100 MHz	W	3.624	3.612	3.636	3.642	3.654	3.660	3.720
	MSs/W	16	33	64	91	135	153	208
143 MHz	W	3.660	3.648	3.660	3.696	3.714	3.720	3.768
	MSs/W	23	49	96	133	197	224	304

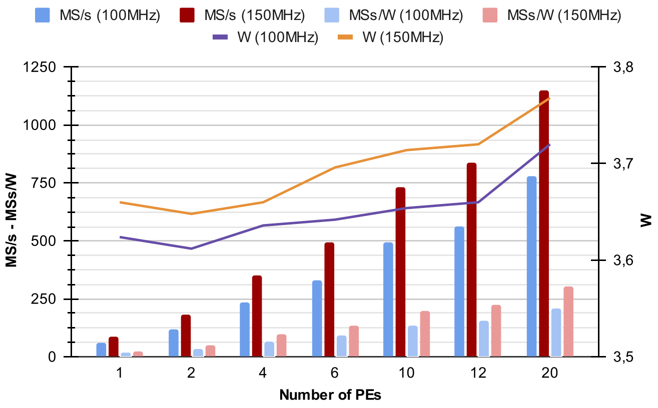

Fig. 7. Performance and power tradeoff analysis of HyperLCA algorithm with adaptive distortion feature in its versions 100 and 143 MHz on ZedBoard (XC7Z020-CLG484) through the current sense (J21 connector).

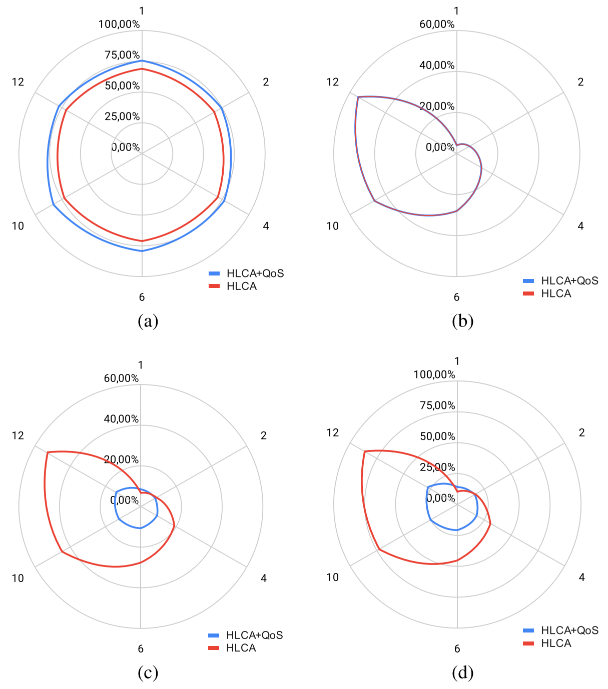

Fig. 8. Comparison of hardware resource utilization in HyperLCA algorithm with and without adaptive distortion feature. (a) BRAM18 K. (b) DSP48E. (c) Flip-flops. (d) LUTs.

power reports. Unfortunately, the ZedBoard platform, which includes XC7Z020-CLG484 as a PL part, does not have multiple power sources to obtain real-time information about the power consumption, so a fine-grain measurement cannot be performed, such as AMD-Xilinx power estimation tool does. On the other hand, the Vivado hardware manager tool monitors the supply voltage of PS and PL parts, but the refresh rate is too low. Therefore, the power consumption of the architecture presented in this article has been measured in the ZedBoard platform across the on-board current sense port (J21 connector), using a multimeter with enough resolution to show the difference in power consumption when a compression process is performed [35]. Table V lists the power consumption in W and energy efficiency in MSs/W in the different versions of the HyperLCA with distortion feature. To calculate the current power consumption, the multimeter has been connected to the J21 connector to measure the voltage across a 10 mΩ resistor. Thus, the power input to the board is calculated with the following equations, where V_{j21} is the voltage measured in the J21 connector, R is 10 mΩ, and V_{in} is the input supply voltage ($V_{\text{in}} = 12$ V):

$$I = V_{j21}/R \tag{10}$$

$$P(\text{W}) = I \times V_{\text{in}}. \tag{11}$$

Fig. 7 graphically shows a comparison of performance and power-consumption tradeoff between the 100 MHz (blue bars and purple line) and 143 MHz (red bars and orange line) of the HyperLCA algorithm with adaptive distortion feature. The power consumption has a similar behavior in both versions, whose variations depend on the number of PEs working in parallel, while energy efficiency in the 143 MHz version is 30% higher than 100 MHz version. On the other hand, the efficiency variation with respect to the data collected in Table IV, where only the FPGA power consumption (PS + PL) is taken into account, has been reduced by 35% and 31% for the 100 MHz and 143 MHz versions, respectively.

V. DISCUSSION

The HyperLCA algorithm with adaptive distortion feature has been compared with the previous implementation published in [8]. Both hyperspectral lossy compressors have been implemented on reconfigurable hardware, concretely in a ZC7Z020-CLG484 FPGA device. Thus, the performance and hardware resource utilization of the two versions are compared and analyzed, as well as the information extracted by each implementation in terms of size and quality. In addition, the hyperspectral lossy compressor presented in this article has been compared with other state-of-the-art transform-based compressors in which reconfigurable devices have been selected to deal with the experimental results of the proposals.

A. Comparison With the Previous Hardware Implementation of the HyperLCA Lossy Compressor

Fig. 8 graphically shows the hardware resource utilization of the HyperLCA algorithm with (HLCA+QoS) and without (HLCA) adaptive distortion feature. The number of BRAMs increases slightly in this new version of the algorithm because the pattern analysis stage introduces new internal memories [see

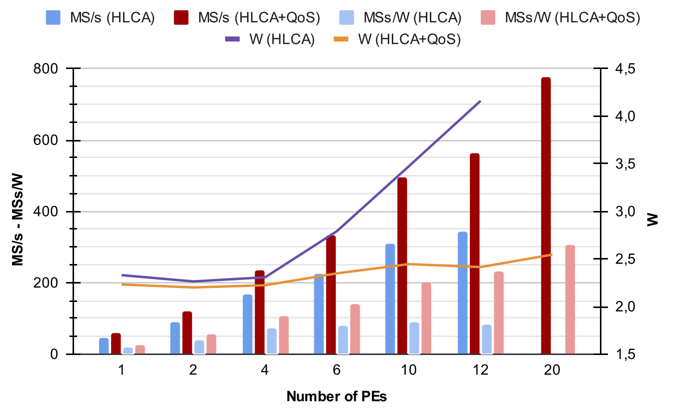

Fig. 9. Performance and power tradeoff analysis of HyperLCA algorithm with and without adaptive distortion feature (100 MHz).

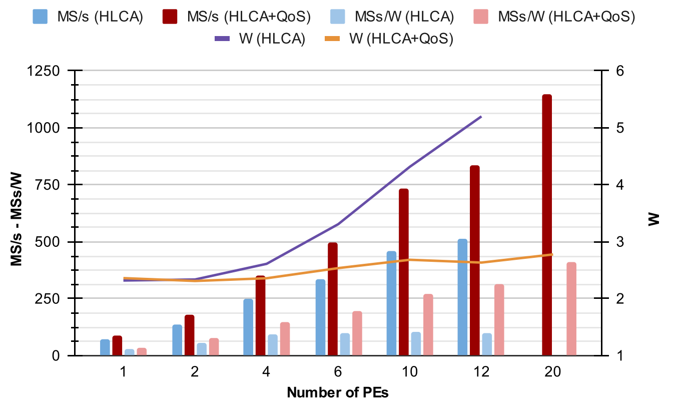

Fig. 10. Performance and power tradeoff analysis of HyperLCA algorithm with and without adaptive distortion feature (143 MHz).

TABLE VI
DEFINED RULES FOR EXPERIMENTAL RESULTS PURPOSE

N_{iter}	$Q_{\max}$ (12)	$Q_{\min}$ (7)	Q_{med} (9)
Rule 1	$T_c \geq 100$	$T_c \leq 50$	otherwise
Rule 2	$T_c \geq 800$	$T_c \leq 200$	otherwise
Rule 3	$T_c \geq 800$	$T_c \leq 600$	otherwise

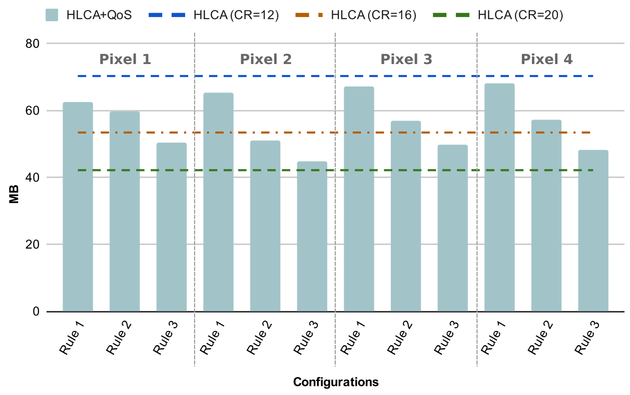

Fig. 11. Compression data size of HyperLCA algorithm with and without adaptive distortion feature with different configurations.

Fig. 8(a)] while the number of DSPs is kept [see Fig. 8(b)]. However, flip-flops and LUTs are considerably reduced by some optimizations performed; the main optimization carried out is the manual partitioning of the orthogonal vectors, $\mathbf{u}$ and $\mathbf{q}$, by a factor equal to the number of PEs that contain the hardware accelerator. For the sake of clarity, the FFs and LUTs are the limiting factors that prevent the deployment of PEs in the previous version of the hardware accelerator, rather than DSPs, which are the limiting factor in this new version. Thus, the best configuration of the HyperLCA with distortion feature, i.e., the 20 PEs version, is not considered in the hardware resource utilization comparison because it cannot be compared.

The performance and power tradeoff achieved by the HyperLCA with and without adaptive distortion feature are shown in Figs. 9 and 10, where the clock frequency has been configured at 100 MHz and 143 MHz, respectively. The new version gets better MSs/W, and it is doubled in the configurations that instantiate 10 and 12 PEs; the light blue bars of Figs. 9 and 10 show the MSs/W achieved by the previous implementation of the HyperLCA, while the light red bars of the aforementioned figures represent the obtained MSs/W by the implementation presented in this article. The other configurations slightly improve the MSamples/s achieved (see blue and red bars of Figs. 9 and 10). In turn, the performed optimizations also reduce the power consumption of the hardware accelerator with respect to the previous version, which is more evident for cases with a higher number of instantiated PEs; purple and orange lines of Figs. 9 and 10 draw the watts of the hardware accelerator with and without the adaptive distortion feature, respectively.

The previous implementation of the hyperspectral lossy compressor extracts the identical amount of spectral information from each block that composes the hyperspectral image, so the applied compression ratio is linear and is defined at design time. It means that the size of extracted data is equal for hyperspectral images with the same spatial and spectral values and the quality performance of compression is kept. Nevertheless, the adaptive distortion feature introduces a nonlinear behavior in terms of quality performance and compression ratio, as well as preserves the linearity feature of transform-based algorithms, i.e., the set of core operations applied to extract the spectral information has a linear behavior. To do so, the number of iterations performed on the set of core operations is set by the predefined rules that depend on the number of pixels close to the hyperspectral pattern signature. For experimental results, the three rules, listed in Table VI, have been defined to be applied to the sensed images; meanwhile, the pattern signatures have been selected from a previous flight, where the weather was similar to that of the compression flight.

Fig. 11 graphically shows the compression data size after the compression process of a hyperspectral image of 377.5 MB using the previous implementation of the HyperLCA algorithm and the modified one. The hyperspectral cube is composed of 1024 samples, 1024 lines, and 180 bands. The compression ratio of the original implementation of the HyperLCA lossy compressor has been set at design time with the following values: CR = 12, CR = 16, and CR = 20, which match with blue, orange, and green dotted lines of Fig. 11, respectively. On the other hand, the blue bars of Fig. 11 illustrate the nonlinear behavior of the spectral extraction in the HyperLCA lossy compressor with adaptive distortion in which four hyperspectral pixels have been selected as signature patterns to obtain the number of iterations to be applied for each sample of the image according to the rules defined in Table VI. The number of iterations (N_{iter}) has been set to 12, 9, and 7 for the high, medium, and low quality of service, respectively. These values match with the number of iterations performed by the original implementation of the HyperLCA algorithm in the three aforementioned CR configurations, but they can be modified to obtain better compression quality performance.

It is worth mentioning that each hyperspectral block, $\mathbf{M}_k$, contains a header block (see Fig. 4), so a small-size overhead is included. This overhead is actually 8-bits per block, i.e., 1 kB is the overhead introduced by the proposed solution for the images used because they contain 1024 lines. Therefore, the maximum and minimum size of the compressed data are close to blue and green dotted lines of Fig. 11. Since the compression size depends on the two aforementioned factors.

The compression performance of the HyperLCA with adaptive distortion has been evaluated by compressing/decompressing a hyperspectral image with four pattern signatures and three rules (see Table VI). The spectral information lost after the compression process has been analyzed using three conventional metrics: the *SNR*, the *RMSE,* and the *MAD*, which are shown in (12)–(14), respectively. The *SNR* and the *RMSE* provide the average information lost in the compression process, where high values of the *SNR* metric are signs of better compression performance. Meanwhile, higher *RMSE* values mean the compression process is not accurate and introduces large information losses. Finally, the *MAD* metric depicts the amount of lost information for the worst reconstructed image value, where 4096 (2^{12}) is the worst possible value

$$\text{SNR} = 10 \cdot \log_{10}\left(\frac{\sum_{i=1}^{nb}\sum_{j=1}^{np}(I_{i,j})^2}{\sum_{i=1}^{nb}\sum_{j=1}^{np}(I_{i,j} - Ic_{i,j})^2}\right) \quad (12)$$

$$\text{RMSE} = \frac{1}{np \cdot nb} \cdot \sqrt{\sum_{i=1}^{nb}\sum_{j=1}^{np}(I_{i,j} - Ic_{i,j})^2} \quad (13)$$

$$\text{MAD} = \max(I_{i,j} - Ic_{i,j}). \quad (14)$$

Table VII presents the average results obtained for the HyperLCA compressor in the three first rows for CR = 12, CR = 16, and CR = 20, whilse the rest of the rows are the result obtained by the modified version of the algorithm, i.e., including the adaptive distortion feature. These results draw different conclusions that should be considered. First, the results are between the lowest and highest configurations of the original implementation because the QoS has been defined with the same number of iterations that the configuration set for the implementation without distortion feature. In addition, it is confirmed that the proposed solution provides better quality compression results than the ones obtained by the previous implementation with the highest compression ratio. It is worth mentioning that the compression performance results obtained by the *Pixel 1 - Rule 1* configuration are equal to the CR = 20 configurations, but the spectral information extracted differs because the distortion feature is able to retrieve more spectral information from the areas of interest; hence, the data compression size is lager (see Fig. 11). Furthermore, it can also be concluded that the HyperLCA lossy compressor with adaptive distortion is able to compress the hyperspectral data with high compression ratios and without introducing significant spectral information losses.

TABLE VII
COMPARISON OF THE QUALITY COMPRESSION RESULTS FOR THE HYPERLCA WITH AND WITHOUT ADAPTIVE DISTORTION FEATURE

Version	Configuration		SNR	MAD	MSE
HLCA	$N_{\text{bits}} = 12$	CR = 12	43.01	24.50	3.12
		CR = 16	42.27	34.25	3.40
		CR = 20	41.31	41.25	3.73
HLCA+QoS	Pixel 1	Rule 1	42.54	29.07	3.29
		Rule 2	41.83	36.16	3.54
		Rule 3	41.31	41.25	3.73
	Pixel 2	Rule 1	42.72	27.40	3.22
		Rule 2	41.49	39.51	3.67
		Rule 3	41.88	35.71	3.53
	Pixel 3	Rule 1	42.85	26.11	3.18
		Rule 2	42.21	32.42	3.41
		Rule 3	41.71	37.39	3.59
	Pixel 4	Rule 1	42.89	25.72	3.16
		Rule 2	42.26	31.97	3.39
		Rule 3	41.71	37.39	3.59

B. Comparison With Other State-of-the-Art Compressors

Furthermore, the throughput and hardware resource utilization of the hyperspectral lossy compressor with adaptive distortion feature have been compared with other state-of-the-art compressors also implemented on reconfigurable hardware. The state-of-the-art architectures and the one explained in this article are summarized in Table VIII, where the selected device, the implemented algorithm, the hardware resource utilization, the clock frequency, and the achieved throughput (MSamples/s) are listed for each proposal. In addition, the performance analysis has been expanded in Table IX, where the power consumption is included as well as the relationship between throughput in MB/s and W.

Prediction-based compression algorithms are good candidates to be implemented on reconfigurable hardware because of the suitability of matrix operations on them. Typically, they depend on the correlation between adjacent pixels in hyperspectral data in which the differences between correlated values are encoded with fewer bits than the actual values. Báscones et al. present an architecture based on low-complexity predictive lossy compression (LCPLC), which is an algorithm based on prediction, uniform threshold quantization, and rate–distortion optimization. Thus, the proposed solution implements a pipeline architecture developed in VHDL and deployed on XQR5VFX130 and XC7VX690 T devices in [40] and [41], respectively. The proposed architecture achieves a throughput of up to

TABLE VIII
HARDWARE RESOURCE AND THROUGHPUT COMPARISON WITH OTHER FPGA-BASED IMPLEMENTATIONS OF HYPERSPECTRAL COMPRESSORS

Proposal	**Device**	**Programming method**	**Compression algorithm**	**BRAMs**	**DSPs**	**FFs**	**LUTs**	**Freq.** (MHz)	**Throughput** (MSamples/s)
Santos et al. [36]	XQR5VFX130	HLS	LCE	17 *(02.85%)*	4 *(01.25%)*	4208 *(20.55%)*	7836 *(05.98%)*	80.2	30.25
García et al. [37]	XC5VFX100	HLS	LCE	4 *(00.88%)*	25 *(09.77%)*	1937 *(03.03%)*	7746 *(12.10%)*	86.96	27.7
Keymulen (1 core)[38]	XC7VX690T	VHDL	FLEX (CCSDS)	72 *(02.45%)*	31 *(00.86%)*	9158 *(01.06%)*	13 134 *(03.03%)*	200	9
Keymulen (15 cores)[38]	XC7VX690T	VHDL	FLEX (CCSDS)	1112 *(37.82%)*	465 *(12.92%)*	260 750 *(30.10%)*	297 807 *(34.37%)*	100	95
Fernández et al. [39]	XC7VFX690T	VHDL	PCA	897 *(30.51%)*	2580 *(71.67%)*	57 564 *(06.64%)*	294 454 *(67.99%)*	75.99	80.29
Báscones et al. [40]	XQR5VFX130	VHDL	LCPLC-JYPEC	10 *(01.68%)*	5 *(01.56%)*	-	6837 *(05.22%)*	247.35	119.96
Báscones et al. [41]	XC7VX690T	VHDL	LCPLC-JYPEC	6 *(00.20%)*	5 *(00.14%)*	-	6731 *(01.55%)*	341.99	162.3
Barrios et al. (1 core) [42]	XCZU9EG	HLS	CCSDS	39 *(04.28%)*	5 *(00.20%)*	8639 *(01.58%)*	11 965 *(04.37%)*	100	0.4
Barrios et al. (8 cores)[42]	XCZU9EG	HLS	CCSDS	312 *(34.21%)*	26 *(01.03%)*	42 771 *(07.80%)*	56 800 *(20.72%)*	100	1.7
Caba et al. (12 PEs) [8]	ZC7Z020	VHDL-HLS	HyperLCA	199 *(71.07%)*	122 *(55.45%)*	56 463 *(53.07%)*	46 116 *(86.68%)*	100	342
Caba et al. (12 PEs) [8]	ZC7Z020	VHDL-HLS	HyperLCA	199 *(71.07%)*	122 *(55.45%)*	56 463 *(53.07%)*	46 116 *(86.68%)*	143	511
Our (20 PEs)	ZC7Z020	VHDL-HLS	HyperLCA	218 *(77.86%)*	202 *(91.82%)*	20 028 *(18.82%)*	19 415 *(36.49%)*	100	777
Our (20 PEs)	ZC7Z020	VHDL-HLS	HyperLCA	218 *(77.86%)*	202 *(91.82%)*	20 028 *(18.82%)*	19 415 *(36.49%)*	143	1,149

TABLE IX
PERFORMANCE COMPARISON WITH OTHER FPGA-BASED IMPLEMENTATIONS OF HYPERSPECTRAL COMPRESSORS

Proposal	**Hyperspectral image**	**Samples**	**Lines**	**Bands**	**Throughput** (MSamples/s)	**MB/s**	**Watts**	**MBS/W**
Santos et al. [36]	-	16	16	256	30.25	0.236	2.029*	0.116
García et al. [37]	AVIRIS (Jasper Ridge)	614	512	224	27.7	5.449	2.022*	2.695
Keymulen (1 core)[38]	AVIRIS (Jasper Ridge)	614	512	224	9	4.722	7.84	0.602
Keymulen (15 cores)[38]	AVIRIS (Jasper Ridge)	614	512	224	95	49.842	11.4	4.372
Fernández et al. [39]	AVIRIS (Jasper Ridge)	614	512	224	80.29	9.004	2.46*	3.661
Báscones et al. [40]	AVIRIS	512	512	256	119.96	29.99	2.732	10.977
Báscones et al. [41]	AVIRIS	512	512	256	162.3	40.575	0.714	56.828
Barrios et al. (1 core) [42]	AVIRIS	512	512	256	0.4	0.1	0.6*	0.167
Barrios et al. (8 cores)[42]	AVIRIS	512	512	256	1.7	0.425	0.9*	0.472
Caba et al. (12 PEs) [8]	Own *(Specim FX10)*	1024	1024	180	342	120.234	1.6	75.146
Caba et al. (12 PEs) [8]	Own *(Specim FX10)*	1024	1024	180	511	179.648	2.22	80.923
Our (20 PEs - 100 MHz)	Own *(Specim FX10)*	1024	1024	180	777	273.164	0.993	275.089
Our (20 PEs - 143 MHz)	Own *(Specim FX10)*	1024	1024	180	1,149	403.945	1.225	329.751

*Obtained from XPE datasheet .

162 MSamples/s and a low-power requirement, roughly 0.714 W. Meanwhile, lossy compression for exomars (LCE) is an algorithm designed for on-board image compression for the European Space Agency ExoMars mission. It consists of four phases: prediction, rate–distortion optimization, quantization, and entropy coding using Golomb codes. It has been accelerated by Santos et al. and García et al. in [36] and [37] using reconfigurable hardware. Both solutions have been developed with HLS (CatapultC) by obtaining good ratio between hardware utilization and throughput. Keymulen [38] implements the FLEX algorithm by introducing a feedback branch with an inverse predictor. It can estimate the value of subsequent samples by controlling the image quality with an adaptive filtering defined by the user. The architecture achieves up to 95 MSamples/s when 15 cores are instantiated by increasing $19\times$ when only one core is working; however, the hardware utilization is considerably increased.

Barrios et al. [42] propose the implementation of a lossy extension of CCSDS by adding a quantizer and a bit-rate feedback loop to control the losses and achieve the desired compression ratios without excessively deteriorating the quality of the decompressed image. The algorithm has been implemented using HLS techniques and has been implemented on a reconfigurable scalable architecture (ARTICo), which provides adaptive computing at runtime to increase the number of instantiated cores and, hence, the performance achieved. Thus, the solution working with one core gets 0.4 MSamples/s, while the one instantiating eight cores achieves 1.7 MSamples/s. Although the runtime flexibility of the ARTICo architecture is a value-added feature in space scenarios, the main problem lies in the context switches and in the movement of data to the local memories of each of the eight reconfigurable regions.

The transform-based algorithms are also suitable for reconfigurable hardware. The basic idea behind this class of algorithms is to map the spatial domain of an image into its transformation domain. Then, the coefficients with larger amplitude, or energy, are encoded with fewer codewords than coefficients with low amplitude to obtain higher compression ratios. The transform function is first applied to generate the transform coefficients. Then, the transform coefficients are decorrelated to remove redundancy. Finally, the output coefficients are passed to the entropy encoder to generate the compressed stream. Fernández et al. [39] propose a principal component analysis (PCA) based solution in reconfigurable hardware by using VHDL to carry out the dimensionality reduction in hyperspectral images. Meanwhile, the architecture presented in this article and its previous implementation in [8] are based on transform operations in which a high parallelization of the operations results in the highest performance.

From the point of view of hardware resource utilization, Table VIII lists the FPGA-based resource utilization of each hyperspectral compressor and its percentage according to the selected device. Thus, the percentage of utilization can be used to compare the state-of-the-art architectures in order to avoid misleading interpretations of information, where most solutions use mid or large FPGA devices instead of cost-optimized ones, or where customized embedded resources, such as DSPs, are not properly used to perform mathematical operations. In this sense, our architecture achieves the best throughput of the proposed architectures, up to 1149 MSamples/s; it is mainly due to the instantiation of DSPs to carry out the operations of transformations instead of the use of other unsuitable FPGA resources for such operations. This behavior is also seen in other proposals, such as the one presented by Keymulen [38] or Fernández et al. [39]; the version with 15 cores working in parallel presented by Keymulen achieves higher throughput that the version with a core; it increases the number of DSPs and reduces the clock frequency. It can be concluded that the use of a hybrid solution, VHDL and HLS, obtains a good use of hardware resources and maintains the benefits of both, i.e., the engineering productivity is kept by using HLS to describe some modules/parts of the architecture, while the modules that require a high degree of parallelism are developed with a hardware description language, as well as the FSM to synchronize the parts described with HLS.

Table IX presents a detailed performance summary achieved by the state-of-the-art architectures and the one presented in this work, using the values obtained after postimplementation phase in order to analyze the architectures at the same point. In addition, the table also lists in the second last column the on-chip power depicted in W. Unfortunately, some state-of-the-art proposals do not provide such information, so it has been estimated with the XPE tool. The use of specialized embedded resources of the FPGA does not have a high impact on the power consumption, but it depends on the number of clock regions that are active, the amount of resources, and the FPGA technology used. In this sense, the architecture presented by Keymulen [38] requires more power than the one presented by Báscones et al. [41], where both proposals use the same FPGA device and VHDL to describe the architecture; the difference lies in the number of FPGA resources used for each proposal. We can conclude that the architecture presented in this work beats the other state-of-the-art proposals in the MB/s per watt (MBS/W) tradeoff, where the fastest version of the HyperLCA compressor with distortion feature, i.e., the version that contains 20 PEs working in parallel, is between $4.9\times$ and $5.8\times$ better than the architecture presented by Báscones et al. [41], when the clock frequency is configured at 100 MHz and 143 MHz, respectively.

It is worth mentioning that the previous implementation of the HyperLCA compressor is between $3.6\times$ and $4.1\times$ slower than the version with adaptive distortion feature. In addition, the clock frequency configuration has little influence on the rate of MB/s/W, and it is increased by 54.66 MBs/W when the clock frequency is set to 143 MHz. Therefore, the architecture presented in this work is able to compress the aforementioned hyperspectral image of 1024×1024 spatial size and 180 spectral bands in 0.935 s with a power consumption of 1.145 W.

VI. Conclusion

This work has dealt with the inclusion of the adaptive distortion feature in the HyperLCA algorithm in order to increase the spectral information in those regions of interest, whose spectral information contains an amount of pixels close to a predefined hyperspectral pattern signature. Thus, the proposal can be adapted to the scenario that is being processed, e.g., more spectral information could be collected from the vegetation than from the soil in a vineyard. In addition, the compressor can be configured to increase the information in the lines that contain anomaly pixels, e.g., ships in the middle of the sea.

The set of core hyperspectral operations is inherited from previous works [8], [9], where the suitability of the FPGA technology for this type of application was tested, so a significant amount of time is saved. However, the new architecture includes new optimizations that allow to instantiate more PEs working in parallel. Consequently, the throughput is considerably increased by the optimizations performed, whereas the adaptive distortion feature introduces a small overhead in size terms due to the need to include an 8-bit header for each sample that is compressed. The solution combines HLS and VHDL modules, bringing an efficient dataflow that meets the requirements of an on-board real-time processing with a push broom camera, concretely with the *Specim FX10*.

Furthermore, we provide a comparison with other FPGA-based architectures of the state-of-the-art in which the conclusions learned from the discussion reveal that the proposed architecture is a cost–energy-efficient solution without reducing the compression quality when the number of transform-based operations carried out is between the ones defined in the previous implementation. Moreover, the loss spectral information can be reduced by increasing the parameter $p_{\max}$, so better compression quality can be achieved.

References

[1] A. Plaza et al., “Recent advances in techniques for hyperspectral image processing,” *Remote Sens. Environ.*, vol. 113, pp. S110–S122, 2009.

[2] A. Altamimi and B. B. Youssef, “A systematic review of hardware-accelerated compression of remotely sensed hyperspectral images,” *Sensors*, vol. 22, no. 1, 2022, Art. no. 263.

[3] M. Radosavljević et al., “Lossy compression of multispectral satellite images with application to crop thematic mapping: A HEVC comparative study,” *Remote Sens.*, vol. 12, no. 10, 2020, Art. no. 1590.

[4] F. Ortenberg, P. Thenkabail, J. Lyon, and A. Huete, “Hyperspectral sensor characteristics: Airborne, spaceborne, hand-held, and truck-mounted; integration of hyperspectral data with LIDAR,” in *Hyperspectral Remote Sensing of Vegetation*. Boca Raton, FL, USA: CRC Press, 2011, pp. 39–68.

[5] J. M. Melián et al., “Real-time hyperspectral data transmission for UAV-based acquisition platforms,” *Remote Sens.*, vol. 13, no. 5, 2021, Art. no. 850.

[6] E. Morin, M. Maman, R. Guizzetti, and A. Duda, “Comparison of the device lifetime in wireless networks for the Internet of Things,” *IEEE Access*, vol. 5, pp. 7097–7114, 2017.

[7] R. Guerra, E. Martel, J. Khan, S. López, P. Athanas, and R. Sarmiento, “On the evaluation of different high-performance computing platforms for hyperspectral imaging: An OpenCL-based approach,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 10, no. 11, pp. 4879–4897, Nov. 2017.

[8] J. Caba, M. Díaz, J. Barba, R. Guerra, and J. A. de la Torre, and S. López, “FPGA-based on-board hyperspectral imaging compression: Benchmarking performance and energy efficiency against GPU implementations,” *Remote Sens.*, vol. 12, no. 22, 2020, Art. no. 3741. [Online]. Available: https://www.mdpi.com/2072-4292/12/22/3741

[9] R. Guerra, Y. Barrios, M. Díaz, A. Baez, S. López, and R. Sarmiento, “A hardware-friendly hyperspectral lossy compressor for next-generation space-grade field programmable gate arrays,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 12, pp. 4813–4828, Dec. 2019.

[10] G. Benelli, G. Meoni, and L. Fanucci, “A low power keyword spotting algorithm for memory constrained embedded systems,” in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr.*, 2018, pp. 267–272.

[11] T. Yan, N. Zhang, J. Li, W. Liu, and H. Chen, “Automatic deployment of convolutional neural networks on FPGA for spaceborne remote sensing application,” *Remote Sens.*, vol. 14, no. 13, 2022, Art. no. 3130.

[12] M. Xu, L. Chen, H. Shi, Z. Yang, J. Li, and T. Long, “FPGA-based implementation of ship detection for satellite on-board processing,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 15, pp. 9733–9745, Oct. 2022.

[13] S. Liu and W. Luk, “Towards an efficient accelerator for DNN-based remote sensing image segmentation on FPGAs,” in *Proc. 29th Int. Conf. Field Programmable Log. Appl.*, 2019, pp. 187–193.

[14] G. Furano et al., “Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities,” *IEEE Aerosp. Electron. Syst. Mag.*, vol. 35, no. 12, pp. 44–56, Dec. 2020.

[15] L. Li, S. Zhang, and J. Wu, “Efficient object detection framework and hardware architecture for remote sensing images,” *Remote Sens.*, vol. 11, no. 20, 2019, Art. no. 2376.

[16] Intel Movidius, VPU intel movidius myriad X, Accessed on: Mar., 10 2023. [Online]. Available: https://www.intel.es/content/www/es/es/products/sku/204770/intel-movidius-myriad-x-vision-processing-unit-0gb/specifications.html

[17] Coral.ia, “Coral dev board datasheet,” Accessed on: Mar. 10, 2023. [Online]. Available: https://coral.ai/docs/dev-board/datasheet/#system-components

[18] V. Kothari, E. Liberis, and N. D. Lane, “The final frontier: Deep learning in space,” in *Proc. 21st Int. Workshop Mobile Comput. Syst. Appl.*, 2020, pp. 45–49.

[19] E. Rapuano et al., “An FPGA-based hardware accelerator for cnns inference on board satellites: Benchmarking with myriad 2-based solution for the cloudscout case study,” *Remote Sens.*, vol. 13, no. 8, 2021, Art. no. 1518.

[20] B. Huang, *Satellite Data Compression*. New York, NY, USA: Springer, 2011.

[21] Consultative Committee for Space Data Systems (CCSDS), “Image data compression. CCSDS, Green Book 120.1-G-3,” Accessed on: Sep., 29 2022. [Online]. Available: https://public.ccsds.org/Pubs/120x1g3.pdf

[22] Y. Dua, V. Kumar, and R. S. Singh, “Comprehensive review of hyperspectral image compression algorithms,” *Opt. Eng.*, vol. 59, no. 9, 2020, Art. no. 090902.

[23] A. B. Kiely et al., “The new CCSDS standard for low-complexity lossless and near-lossless multispectral and hyperspectral image compression,” *Nat. Aeronaut. Space Admin.*, no. 2, Washington, DC, USA, Dec. 2022.

[24] E. Augé, J. E. Sánchez, A. Kiely, I. Blanes, and J. Serra-Sagrista, “Performance impact of parameter tuning on the CCSDS-123 lossless multi- and hyperspectral image compression standard,” *J. Appl. Remote Sens.*, vol. 7, no. 1, 2013, Art. no. 074594.

[25] L. Santos, E. Magli, R. Vitulli, J. F. López, and R. Sarmiento, “Highly-parallel GPU architecture for lossy hyperspectral image compression,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 6, no. 2, pp. 670–681, Apr. 2013.

[26] Y. Barrios, A. J. Sánchez, L. Santos, and R. Sarmiento, “SHyLoC 2.0: A versatile hardware solution for on-board data and hyperspectral image compression on future space missions,” *IEEE Access*, vol. 8, pp. 54269–54287, 2020.

[27] R. Guerra, Y. Barrios, M. Díaz, L. Santos, S. López, and R. Sarmiento, “A new algorithm for the on-board compression of hyperspectral images,” *Remote Sens.*, vol. 10, no. 3, 2018, Art. no. 428.

[28] M. Díaz et al., “Real-time hyperspectral image compression onto embedded GPUs,” *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 8, pp. 2792–2809, Aug. 2019.

[29] I. Vasilyeva, F. Li, S. Abramov, V. V. Lukin, B. Vozel, and K. Chehdi, “Lossy compression of three-channel remote sensing images with controllable quality,” in *Proc. Image Signal Process. Remote Sens. XXVII*, 2021, pp. 205–216.

[30] P. G. Howard and J. S. Vitter, “Fast and efficient lossless image compression,” in *Proc. Data Compression Conf.*, 1993, pp. 351–360.

[31] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, “High-level synthesis for FPGAs: From prototyping to deployment,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.

[32] P. Horstrand, R. Guerra, A. Rodríguez, M. Díaz, S. López, and J. F. López, “A UAV platform based on a hyperspectral sensor for image capturing and on-board processing,” *IEEE Access*, vol. 7, pp. 66919–66938, 2019.

[33] Specim, *Spectral Imaging, Ltd.*, “Specim fx10 datasheet,” Accessed on: Dec. 1, 2022. [Online]. Available: https://www.specim.fi/wp-content/uploads/2020/02/Specim-FX10-Technical-Datasheet-04.pdf

[34] AMD-Xilinx, “Seven steps to an accurate worst-case power analysis using the Xilinx power estimator,” Accessed on: Apr., 15 2023. [Online]. Available: https://docs.xilinx.com/v/u/en-US/xapp1348-power-analysis

[35] Velleman, “User manual: DVM1200—Multimeter with USB interface,” Accessed on: May, 26 2023. [Online]. Available: https://www.velleman.eu/downloads/1/dvm1200gbnlfresdplit.pdf

[36] L. Santos, J. F. López, R. Sarmiento, and R. Vitulli, “FPGA implementation of a lossy compression algorithm for hyperspectral images with a high-level synthesis tool,” in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2013, pp. 107–114.

[37] A. García, L. Santos, S. López, G. Marrero, J. F. López, and R. Sarmiento, “High level modular implementation of a lossy hyperspectral image compression algorithm on a FPGA,” in *Proc. 5th Workshop Hyperspectral Image Signal Process., Evol. Remote Sens.*, 2013, pp. 1–4.

[38] D. Keymeulen, “FPGA implementation of lossless and lossy compression of space-based multispectral and hyperspectral imagery,” *Mil. Aerosp. Programmable Logic Devices (MAPLD) Workshop*, La Jolla, CA, 2016.

[39] D. Fernández, C. González, D. Mozos, and S. Lopez, “FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images,” *J. Real-Time Image Process.*, vol. 16, pp. 1395–1406, 2019.

[40] D. Báscones, C. González, and D. Mozos, “An extremely pipelined FPGA implementation of a lossy hyperspectral image compression algorithm,” *IEEE Trans. Geosci. Remote Sens.*, vol. 58, no. 10, pp. 7435–7447, Oct. 2020.

[41] D. Báscones, C. González, and D. Mozos, “An FPGA accelerator for real-time lossy compression of hyperspectral images,” *Remote Sens.*, vol. 12, no. 16, 2020, Art. no. 2563.

[42] Y. Barrios et al., “Lossy hyperspectral image compression on a reconfigurable and fault-tolerant FPGA-based adaptive computing platform,” *Electronics*, vol. 9, no. 10, 2020, Art. no. 1576.

Julián Caba received the M.S. degree in computer science and the Ph.D. degree in computer science from the University of Castilla-La Mancha (UCLM), Ciudad Real, Spain, in 2009 and 2018, respectively.

He is currently an Assistant Professor with TSI Department, UCLM. His current research interests include hardware verification methodologies, embedded systems, high-level synthesis, runtime reconfigurable systems, and heterogeneous distributed systems.

Dr. Caba was a recipient of the Ph.D. category in the Xilinx Open Hardware Contest in 2017.

Dirk Stroobandt (Member, IEEE) received the Ph.D. degree in electrotechnical engineering from Ghent University, Ghent, Belgium, in 1998.

He was a Visiting Researcher with the University of California at Irvine, Irvine, CA, USA, in 1997, and the University of California at Los Angeles, Los Angeles, CA, USA, from 1999 to 2000. He is currently a Full Professor with Computer Systems Laboratory, Department of Electronics and Information Systems, Ghent University, where he also leads the research group Hardware and Embedded Systems with interests in semiautomatic hardware design, runtime field-programmable gate array reconfiguration, and reconfigurable multiprocessor networks.

María Díaz was born in Spain, in 1990. She received the Industrial Engineering degree from the University of Las Palmas de Gran Canaria (ULPGC), Las Palmas de Gran Canaria, Spain, in 2014, the master's degree in system and control engineering jointly from the *Universidad Complutense de Madrid* and the *Universidad Nacional de Educacin a Distancia*, Madrid, Spain, in 2017 and the Ph.D. degree in telecommunication technologies and computer engineering from the ULPGC, Las Palmas de Gran Canaria, Spain, in 2021.

She developed her research activities with Integrated Systems Design Division, Institute for Applied Microelectronics, ULPGC. Additionally, she conducted a research stay with GIPSA-Lab, University of Grenoble Alpes, France, and the University of Castilla-La Mancha, Spain. Her research interests include image and video processing, development of highly parallelized algorithms for hyperspectral image processing, and hardware implementation.

Jesús Barba received the M.S. and Ph.D. degrees in computer engineering diploma from the University of Castilla-La Mancha (UCLM), Ciudad Real, Spain, in 2001 and 2008, respectively.

He has been an Associate Professor with the Department of Information and Systems Technology (TSI) since 2001 and a member of the ARCO research group, located at the School of Computer Science, UCLM, Spain. Among the open research lines and interests, it is worth mentioning the following: low-cost and low-power reconfigurable systems for ubiquitous computing, reconfigurable computing platforms for AI algorithms, heterogeneous distributed embedded computing, and high-level synthesis tools.

Fernando Rincón (Member, IEEE) received the degree in computer science from the Autonomous University of Barcelona, Barcelona, Spain, in 1993, and the Ph.D. degree from the University of Castilla-La Mancha, Ciudad Real, Spain in 2003.

He is currently an Assistant Professor with TSI Department, University of Castilla-La Mancha. His research interests include system-on-chip integration, HW runtime reconfiguration, and heterogeneous distributed systems.

Sebastián López (Member, IEEE) was born in Las Palmas de Gran Canaria, Spain, in 1978. He received the Electronic Engineering degree from the University of La Laguna, San Cristobal de La Laguna, Spain, in 2001, and the Ph.D. degree in electronic engineering from the University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, in 2006.

He is currently an Associate Professor with the University of Las Palmas de Gran Canaria, where he is involved in research activities with Integrated Systems Design Division, Institute for Applied Microelectronics. He has coauthored more than 120 articles in international journals and conferences. His research interests include real-time hyperspectral imaging, reconfigurable architectures, high-performance computing systems, and image and video processing and compression.

Dr. López was a recipient of regional and national awards during his electronic engineering degree. He also serves as an Active Reviewer for different JCR journals and as a Program Committee Member of a variety of reputed international conferences. Furthermore, he acted as one of the program chairs of the IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing in its 2014 edition and of the SPIE Conference of High-Performance Computing in Remote Sensing, from 2015 to 2018. He is also an Associate Editor for the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING, *MDPI Remote Sensing*, and *Mathematical Problems in Engineering Journal*. He was an Associate Editor for the IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, from 2008 to 2013. Moreover, he has been the Guest Editor of different special issues in JCR journals related to his research interests.

Juan Carlos López (Member, IEEE) received the M.S. and Ph.D. degrees in telecommunication (electrical) engineering from the Technical University of Madrid, Madrid, Spain, in 1985 and 1989, respectively.

From 1989 to 1999, he was an Associate Professor with the Department of Electrical Engineering, Technical University of Madrid. From September 1990 to August 1992, he was a Visiting Scientist with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. From 2000 to 2008, he was the Dean of the School of Computer Science, University of Castilla-La Mancha. He is currently a Professor of computer architecture with the University of Castilla-La Mancha, Ciudad Real, Spain. His research interests include embedded system design, distributed computing, and advanced communication services.

Dr. Lopez is a member of the IEEE and ACM. He is and has been a member of different panels of the Spanish National Science Foundation and the Spanish Ministry of Education and Science, regarding the information technologies research programs.