INVITED PAPER

The Promise of Reconfigurable Computing for Hyperspectral Imaging Onboard Systems: A Review and Trends

Fast processing solutions for compression and/or interpretation of hyperspectral data onboard spacecraft imaging platforms are discussed in this paper with the purpose of giving a more efficient exploitation of hyperspectral data sets in various applications.

By SEBASTIAN LOPEZ, *Member IEEE*, TANYA VLADIMIROVA, *Member IEEE*, CARLOS GONZÁLEZ, JAVIER RESANO, DANIEL MOZOS, AND ANTONIO PLAZA, *Senior Member IEEE*

ABSTRACT | Hyperspectral imaging is an important technique in remote sensing which is characterized by high spectral resolutions. With the advent of new hyperspectral remote sensing missions and their increased temporal resolutions, the availability and dimensionality of hyperspectral data is continuously increasing. This demands fast processing solutions that can be used to compress and/or interpret hyperspectral data onboard spacecraft imaging platforms in order to reduce downlink connection requirements and perform a more efficient exploitation of hyperspectral data sets in various applications. Over the last few years, reconfigurable hardware solutions such as field-programmable gate arrays (FPGAs) have been consolidated as the standard choice for onboard remote sensing processing due to their smaller size, weight, and power consumption when compared with other high-performance computing systems, as well as to the availability of more FPGAs with increased tolerance to ionizing radiation in space. Although there have been many literature sources on the use of FPGAs in remote sensing in general and in hyperspectral remote sensing in particular, there is no specific reference discussing the state-of-the-art and future trends of applying this flexible and dynamic technology to such missions. In this work, a necessary first step in this direction is taken by providing an extensive review and discussion of the (current and future) capabilities of reconfigurable hardware and FPGAs in the context of hyperspectral remote sensing missions. The review covers both technological aspects of FPGA hardware and implementation issues, providing two specific case studies in which FPGAs are successfully used to improve the compression and interpretation (through spectral unmixing concepts) of remotely sensed hyperspectral data. Based on the two considered case studies, we also highlight the major challenges to be addressed in the near future in this emerging and fast growing research area.

KEYWORDS | Field-programmable gate arrays (FPGAs); hyperspectral data compression; hyperspectral remote sensing; reconfigurable hardware; spectral unmixing

Manuscript received October 8, 2012; revised November 27, 2012; accepted November 27, 2012. Date of publication February 5, 2013; date of current version February 14, 2013. This paper was supported by the European Commission in the framework of the TOLOMEO (FP7-PEOPLE-2010-IRSES) project and by the Spanish Government in the framework of the CEOS-SPAIN (AYA2011-29334-C02-02), DREAMS (TEC2011-28666-C04-04), AYA2009-13300-C03-02, TIN2009-09806, and TIN2010-21291-C02-01 projects.

S. Lopez is with the Institute for Applied Microelectronics (IUMA), University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria 35017, Spain (e-mail: seblopez@iuma.ulpgc.es).
T. Vladimirova is with the Department of Engineering, University of Leicester, Leicester LE1 7RH, U.K. (e-mail: t.vladimirova@le.ac.uk).
C. González and **D. Mozos** are with the Department of Computer Architecture and Automatics, Complutense University of Madrid, Madrid 28040, Spain (e-mail: carlosgonzalez@fdi.ucm.es; mozos@fis.ucm.es).
J. Resano is with the Department of Computer and Systems Engineering (DIIS), University of Zaragoza, Zaragoza 50009, Spain (e-mail: jresano@unizar.es).
A. Plaza is with the Hyperspectral Computing Laboratory (HyperComp), Department of Technology of Computers and Communications, University of Extremadura, Caceres 10003, Spain (e-mail: aplaza@unex.es).

Digital Object Identifier: 10.1109/JPROC.2012.2231391

I. INTRODUCTION

Hyperspectral sensors are capable of generating very high-dimensional imagery through the use of sensor optics with a large number of (nearly contiguous) spectral bands, providing very detailed information about the sensed scene. From a remote sensing perspective, the spatial and significantly improved spectral resolutions provided by these latest generation instruments have opened cutting-edge possibilities in many applications, including environmental modeling and assessment, target detection and identification for military and defense/security purposes, agriculture, urban planning and management studies, risk/hazard prevention and response including wild land fire tracking, biological threat detection, and monitoring of oil spills and other types of chemical contamination, among many others.

Because of their potential, remote sensing hyperspectral sensors have been incorporated in different satellite missions over recent years like the currently operating Hyperion on NASA's Earth Observing-1 (EO-1) satellite[1] or CHRIS sensor on the European Space Agency (ESA)'s Proba-1. Furthermore, the remote sensing hyperspectral sensors that will be allocated in future missions will have enhanced spatial, spectral, and temporal resolutions, which will allow capturing more hyperspectral cubes per second with much more information per cube. For example, it has been estimated by the NASA's Jet Propulsion Laboratory (JPL) that a volume of 1–5 TB of data will be daily produced by short-term future hyperspectral missions like the NASA's HyspIRI.[2] Similar data volume ratios are expected in European missions such as Germany's EnMAP[3] or Italy's PRISMA.[4] Unfortunately, this extraordinary amount of information jeopardizes the use of these last-generation hyperspectral instruments in real-time or near-real-time applications, due to the prohibitive delays in the delivery of Earth Observation payload data to ground processing facilities. In this respect, ESA has already flagged up in 2011 that "data rates and data volumes produced by payloads continue to increase, while the available downlink bandwidth to ground stations is comparatively stable" [1]. In this context, the design of solutions that enable to take advantage of the ever increasing dimensionality of remotely sensed hyperspectral images for real-time applications has gained a significant relevance during the last decade.

Within this scenario, onboard processing systems have emerged as an attractive solution in order to decrease the delays between the acquisition of a hyperspectral image, its processing/interpretation, and the decision on a proper action to be taken according to the information extracted from the image. This can be mainly achieved in two ways: 1) performing onboard (lossless or lossy) compression of the acquired data before transmitting them, so that the remotely sensed hyperspectral images are downloaded and further processed at the ground level; and/or 2) processing

[1]http://eo1.gsfc.nasa.gov
[2]http://hyspiri.jpl.nasa.gov
[3]http://www.enmap.org
[4]http://www.asi.it/en/flash_en/observing/prisma

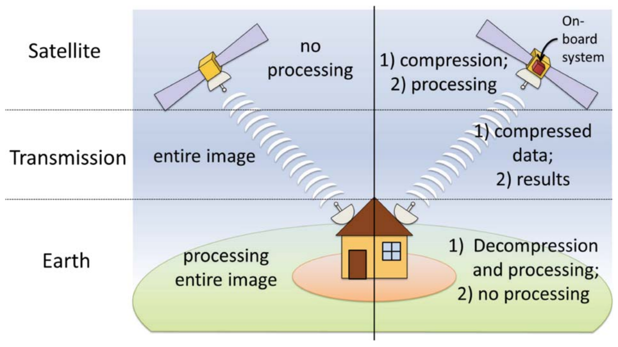

Satellite
no processing
1) compression; 2) processing
On-board system
Transmission
entire image
1) compressed data; 2) results
Earth
processing entire image
1) Decompression and processing; 2) no processing

Fig. 1. ***Onboard processing systems scenario.***

the hyperspectral images according to the needs of an application (or a set of them), so that only the obtained results [i.e., number and location of thematic classes after performing a classification/clustering of the sensed images, location of a set of sought targets within an image, pure spectral signatures (*endmembers*) together with their correspondent abundance factors obtained after spectral unmixing, etc.] are transmitted. At this point, it is worth mentioning that both scenarios, which have been exemplified in Fig. 1 using a toy example, are not mutually exclusive since a possible framework could be to process the image onboard and, then, compress the results obtained prior to the transmission to ground.

For the particular case of Earth Observation satellites, these onboard systems should at least accomplish the following three mandatory characteristics. First, they must allow high computational performance, since all the state-of-the-art algorithms for compressing and/or processing a given hyperspectral image have a huge associated computational burden. Second, they should have compact size and reduced weight and power consumption, due to the inherent nature of remote sensing satellites. Last but not least, they must be resistant to damages or malfunctions caused by ionizing radiation, present in the harsh environment of outer space. Furthermore, it would be highly desirable that these high-performance onboard processing systems could also show a high degree of flexibility so that they can adapt to varying mission needs, faults, and/or to evolving and future processing algorithms and standards.

Among the different general-purpose high-performance computing platforms that are currently commercially available, current radiation-hardened and radiation-tolerant field-programmable gate arrays (FPGAs) undoubtedly represent the best choice in terms of the requirements outlined above, due to their negligible size and mass when compared with traditional cluster-based systems, as well as to their lower power dissipation figures when compared with graphics processing units (GPUs). Because of this reason, this work is focused on demonstrating the suitability of FPGAs to onboard processing of hyperspectral images acquired by current and future remote sensing missions, as well as on highlighting the major challenges to be addressed in the near future. More specifically, in this work, we provide an extensive review of the (current and future) capabilities of FPGAs in the context of remote sensing in general and hyperspectral imaging in particular. The review covers both technological aspects of FPGA hardware and implementation issues, describing two specific case studies in which FPGAs are successfully being used to improve the compression and data interpretation (through spectral unmixing techniques) of remotely sensed hyperspectral data. We believe that this contribution is much needed; although there have been many developments in the literature discussing the use of FPGAs in remote sensing applications, there is no specific reference discussing the state-of-the-art and future trends of applying this flexible and dynamic technology to remote sensing missions.

The remainder of this paper is organized as follows. Section II introduces the basics of FPGA technology, highlighting its most relevant characteristics for onboard systems. Section III reviews the state-of-the-art of FPGA implementations for hyperspectral imaging in general, and for compression and processing of hyperspectral images in particular. The next two sections detail these two example case studies: lossless/lossy compression (Section IV) and spectral unmixing (Section V), going from the architectural level to the FPGA implementation itself and providing implementation examples together with experimental validation and assessment using real hyperspectral scenes collected by a variety of sensors. Finally, Section VI provides some summarizing statements as well as future research hints and challenges.

II. FIELD-PROGRAMMABLE GATE ARRAYS (FPGAs)

An FPGA can be roughly defined as an array of interconnected logic blocks, as is depicted in Fig. 2. One of the main advantages of these devices is that both the logic blocks and their interconnections can be (re)configured by their users as many times as needed in order to implement different combinational or sequential logic functions. In addition, modern FPGAs frequently include embedded hardware modules, such as static RAMs memories or multipliers specifically developed for digital signal processing computations, which can improve the efficiency of the system. As is seen in Fig. 3, this characteristic provides FPGAs with the advantages of both software and hardware systems in the sense that FPGAs exhibit more flexibility and shorter development times than application-specific integrated circuits (ASICs) but, at the same time, are able to provide much more competent levels of performance, in terms of number of operations per watt, than general purpose processors (GPPs). In any case, these different platforms are not mutually exclusive. On the contrary, manufacturing companies such as Xilinx (San Jose CA, USA) [2] or Altera (San Jose, CA, USA) [3] have developed system-on-a-chip platforms that include ASICs modules, GPPs, and FPGAs. These platforms can be used to develop a tightly integrated hardware/software system were the software running in the processors can take advantage of the hardware modules implemented both on the ASICs and on the FPGA. In these platforms, ASICs provide support for frequently used functions, such as those included in the communication standards, and FPGAs are used to customize the device adding hardware support specifically designed for the target applications.

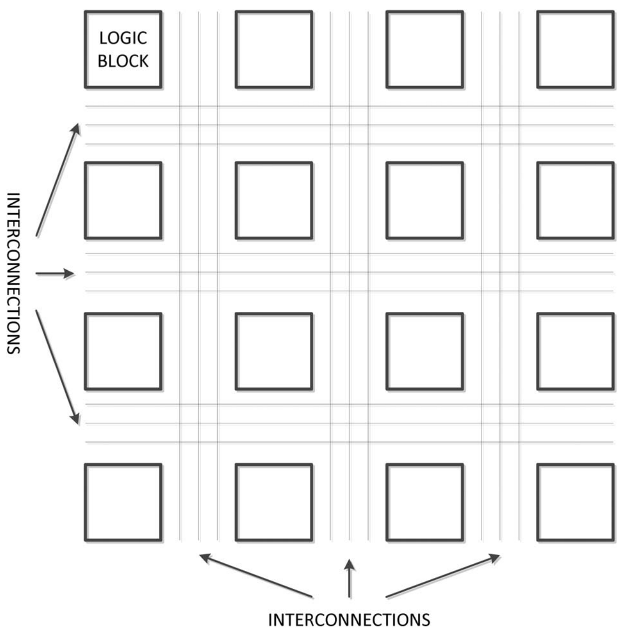


Fig. 2. ***General view of the classical FPGA architecture.***

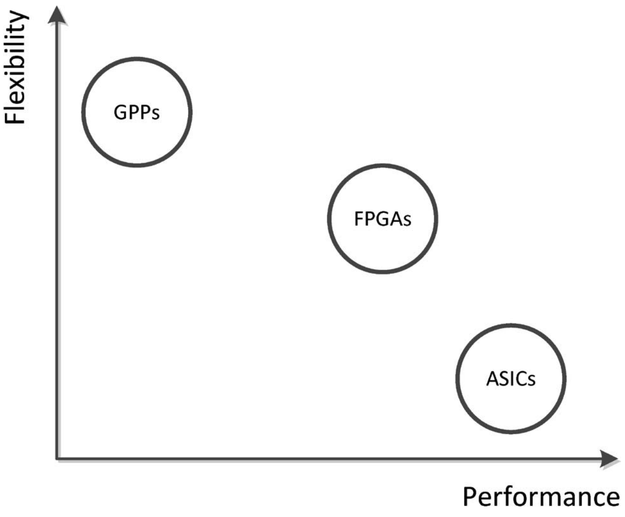


Fig. 3. ***Performance versus flexibility comparison between GPPs, ASICs, and FPGAs.***

FPGAs offer good performance because they can implement optimized data paths for each computational task. These data paths take advantage of the task internal parallelism and include a customized memory hierarchy. Moreover, due to the large number of available resources in current FPGAs, frequently it is also possible to execute several tasks in parallel in order to achieve further speedups. Using optimized data paths not only improves the performance, but also reduces the power and energy consumption when compared with GPPs. The reason is that executing a given task in GPPs involves adjusting the required computations to the GPP instruction set and carrying out an expensive instruction decoding process, which leads to important power and energy overheads. Furthermore, the power and energy efficiency of FPGAs has significantly improved during the last decade. FPGA vendors have achieved this goal improving the FPGA architectures, including optimized hardware modules, and taking advantage of the most recent silicon technology. For instance Xilinx reports a 50% reduction in the power consumption when moving from their previous Xilinx 6 FPGAs (implemented using 40-nm technology) to their most recent Xilinx 7 FPGAs (a new architecture implemented using 28-nm technology).

FPGAs are becoming an increasingly attractive solution for space-based systems not only because of their reduced size, weight, and power dissipation, as well as to their excellent tradeoff behavior between pure software and specific hardware systems, but also because of the following four main reasons: 1) as with terrestrial applications, they perform well in high-throughput signal processing tasks encountered in space, like processing and/or compressing a hyperspectral image; 2) FPGAs permit changes to the usage model and the data processing paradigm in space rather than hard coding of all components prior to launch; in this sense, their inherent ability to change their functionality—through partial or full reconfiguration—aids in their agility, and extends the useful life of remote sensing autonomous systems; 3) when compared to other technologies able to provide similar computational performance and design agility, FPGAs offer clear advantages in terms of size, area, and energy efficiency; at this point, it is important to highlight that FPGAs have demonstrated their superiority in terms of energy efficiency with respect to other popular reduced-area low-weight kinds of general-purpose high-performance devices, such as GPUs; and 4) they can be manufactured in order to resist high levels of radiation without changing the content of their inner memories, which determines the FPGA programming, which undoubtedly constitutes another advantage of FPGAs with respect to current GPUs for space-based systems. Moreover, since FPGAs implement custom designs, additional fault-tolerant levels can be included in the system when needed, such as dual or triple modular redundancy.

Before analyzing the effects of outer space radiation in commercially available FPGAs, it is important to distinguish between antifuse, SRAM (static RAM), and Flash-based FPGAs. In antifuse-based FPGAs, the logical function to be reproduced within the targeted FPGA is obtained by burning off its internal connections, which are made of antifuses. Although these FPGAs are highly tolerant to radiation effects, their inherent nature converts them into onetime programmable devices and, hence, they become less attractive for future space-based systems than SRAM and Flash-based FPGAs, since these last two can be reprogrammed. In a SRAM-based FPGA, the configuration of its logic elements and interconnections is stored in small SRAM memories distributed along the field-programmable array. Since SRAM memories are volatile by default, these FPGAs must be programmed each time they are powered up, the program typically being stored in an off-chip nonvolatile memory. On the contrary, in a Flash-based FPGA, the configuration is stored in a nonvolatile memory and, hence, the FPGA device remains programmed even when the power source is removed from the system.

Besides the fact that each way of (re)programming a FPGA has its own strengthens and weakness (in this sense, interested readers are referred to [4] for a comprehensive analysis), for the scope of this work it is important to highlight that they also behave in different ways under radiation effects. These effects can be categorized into two groups: total ionizing dose (TID) effects and single event effects (SEEs). The former gives an account of the energy accumulated in the device by ionizing radiation per unit of mass, being measured in the international system of units in grays (Gy—joules per kilogram), whereas the latter define the effects caused by aisle ionizing particles, which can be destructive (in the sense that they provoke permanent damages in the device) or not. With independence of its programmability, the families of FPGAs that are up to a certain limit resistant to these radiation effects are known as radiation-tolerant and radiation-hardened FPGAs, and different companies currently offer such devices. For instance, Xilinx presently offers the SRAM-based Virtex-4QV [5] and Virtex-5QV [6] families, with a frequency performance up to 450 MHz, and which have been extensively analyzed in terms of radiation effects by the NASA's Jet Propulsion Laboratory (JPL) [7] while Microsemi manufactures the RT ProASIC3 series [8], based on nonvolatile Flash-based technology, which can run up to 350 MHz.

Different state-of-the-art works have demonstrated that both families of reprogrammable FPGAs are suitable for space-based systems, although SRAM-based FPGAs are in general more tolerant than Flash-based FPGAs to TID effects, while on the contrary, SRAM-based FPGAs are more vulnerable than Flash-based FPGAs to SEEs [9]. In the remainder of this paper, we will center our efforts in demonstrating that both types of FPGAs can be considered for future onboard hyperspectral imaging systems by means of two different case studies: compression and linear unmixing.

III. REVIEW OF STATE-OF-THE-ART FPGA-BASED HYPERSPECTRAL IMAGING SYSTEMS

Recently, significant efforts have been made directed toward the increase of the performance of remote sensing applications, specifically in the acceleration of hyperspectral imaging algorithms [10]–[14]. In the field of hyperspectral remote sensing, mainly four solutions have been addressed: cluster computing, heterogeneous computing, GPUs, and FPGAs. The motivation for this effort relies on the high computation requirements needed for achieving real-time or near-real-time processing capabilities. Due to the superiority of FPGA devices for onboard systems, this section is focused on reviewing the state-of-the-art works that deal with the implementation of hyperspectral imaging algorithms onto FPGAs. More precisely, this section has been divided into three subsections devoted to review the existing FPGA-based implementations for hyperspectral image compression, spectral unmixing, and other hyperspectral imaging algorithms.

A. State-of-the-Art Hyperspectral Image Compression Implementations on FPGAs

Due to the large amount of data generated by sensors, especially spaceborne ones, it is necessary to reduce the size of the data in order to download all the acquired sensor input. Image compression compensates for the limited onboard resources, in terms of mass memory and downlink bandwidth and thus provides a solution to the "bandwidth versus data volume" dilemma of modern spacecraft. The problem becomes increasingly important for hyperspectral remote sensing systems with the current trend toward developing more accurate hyperspectral sensors covering hundreds of spectral bands. Thus, hyperspectral image compression becomes an important onboard capability in order to alleviate the memory costs and communication bottleneck in present and future satellite missions [15].

1) Introduction to Compression of Satellite Imagery: Image compression methods can be divided into two classes: lossless or lossy. With lossless image compression, the reconstructed image is exactly the same as the original one, without any information loss at the expense of achieving relatively modest compression ratios when compared with lossy methods. This is because the entropy, which measures the quantity of information contained in a source, imposes a theoretical boundary on the lossless compression performance, expressed by the lowest compression bit rate per pixel. Entropy depends on the statistical nature of the source and ideally an infinite-order probability model is needed to evaluate it. On the contrary, lossy image compression enables competitive compression ratios at the expense of introducing a varying degree of unrecoverable information loss in the reconstructed images.

The process of lossless compression is achieved via removing the redundancy in the data. There are several types of redundancies in a remotely sensed image, such as spatial redundancy, statistical redundancy, human vision redundancy, and spectral redundancy.

Spatial or intraband redundancy means that the pixel information could be partially deduced by neighboring pixels. Spatial decorrelation methods, like prediction or transformation, are usually employed to remove the spatial redundancy. Prediction is used to predict the current pixel value from neighboring pixels. For example, the differential pulse code modulation (DPCM) method is a typical prediction-based technique. On the other hand, transformation is used to transform the image from the spatial domain into another domain, applying, for example, the discrete cosine transform (DCT) or the discrete wavelet transform (DWT) [15].

Statistical redundancy explores the probability of symbols. The basic idea is to assign short codewords to high probability symbols, and long codewords to low probability symbols. Huffman or arithmetic coding are two popular methods to remove statistical redundancy which are usually known as entropy coding methods. Human vision redundancy explores the fact that eyes are not so sensitive to high frequencies. Removing human vision redundancy is normally achieved by quantization, with high-frequency elements being over quantized or even deleted. Spectral or interband redundancy is present in 3-D hyperspectral imagery with a large number of spectral bands in addition to the spatial redundancy present in 2-D digital images, adding a third dimension to compressing 2-D images.

A typical architecture of a 2-D image compression system consists of a spatial decorrelation stage, which is followed by a quantization and entropy coding stages, exploiting the first three redundancies described above. Based on techniques used for spatial decorrelation, compression systems can be divided into prediction, DCT, and DWT-based systems. Prediction-based compression methods used in space missions include DPCM [16], [17], adaptive DPCM [18], consultative committee for space data systems (CCSDS) lossless data compression (CCSDS-LDC) [19], lossless JPEG [20], and JPEG-LS [21]. DCT-based compression methods include JPEG-baseline [20] and specifically designed DCT compression methods. DWT-based compression methods include JPEG2000 [22], embedded zero-tree wavelet (EZW) [23], SPIHT [24], CCSDS image data compression (CCSDS-IDC) [25], and specifically designed DWT compression methods.

The CCSDS has played and will continue playing a significant role in providing efficient onboard compression methods. In May 1997, CCSDS published a recommendation standard for lossless data compression, which is an extended Rice algorithm with added two low-entropy coding options [19]. This recommendation addresses only lossless source coding and is applicable to a large variety of digital data, generated by different types of imaging or nonimaging instruments onboard satellites. The algorithm consists of two separate functional parts: a preprocessor and an adaptive entropy coder. The preprocessor is used to decorrelate a block of J sample data and subsequently map them into symbols suitable for the entropy coding stage. The entropy coding module is a collection of variable-length codes operating in parallel on blocks of J preprocessed samples. Each code is nearly optimal for a particular geometrically distributed source. The coding option achieving the highest compression is selected for transmission, along with an *ID* bit pattern used to identify the option to the decoder. Because a new compression option can be selected for each block, the algorithm can adapt to changing source statistics.

Since 1998, the CCSDS data compression working group has begun to assess the feasibility of establishing an image compression recommendation suitable for space applications, and the CCSDS–IDC Blue Book was finally produced in November 2005 [25]. The compression technique described in this recommendation can be used to produce both lossy and lossless compression. It supports both frame-based input formats produced, for example, by charge-coupled device (CCD) arrays, and strip-based input formats produced by push-broom-type sensors. An image pixel resolution of up to 16 b is supported. The compressor consists of two functional parts: a DWT module that performs decorrelation and a bit-plane encoder (BPE), which encodes the decorrelated data. Although similar to that of JPEG2000 standard, this architecture differs from it in several respects: 1) it specifically targets high-rate instruments used onboard space missions; 2) a tradeoff is performed between compression performance and complexity with a particular emphasis on space applications; 3) the lower computational complexity of the CCSDS–IDC algorithm supports a fast and low-power hardware implementation; and 4) it has a limited set of options, enabling its successful application without an in-depth algorithm knowledge. According to literature sources, CCSDS–IDC could achieve performance similar to that of JPEG2000 [26]–[29].

2) Basics of Hyperspectral Image Compression: When compressing hyperspectral images, both the spectral and spatial types of redundancy need to be removed in order to achieve a good compression performance via a lossless process or a lossy process. In most of the cases, the spectral decorrelation is performed first, followed by the spatial decorrelation, which is used in 2-D image compression, as depicted in Fig. 4. The spectral decorrelation aims to reduce the spectral redundancy that exists between bands, whereas spatial decorrelation takes care of the interpixel redundancy within a band. This scheme has been used widely in remote sensing as well as in medical applications for 3-D medical data.

Each of the spectral and spatial decorrelation processes can be performed by either a lossless algorithm or a lossy algorithm as shown in Fig. 4. To achieve an overall lossless compression process, both spectral and spatial decorrelation stages should utilize lossless transformations. On the other hand, an overall lossy compression process can utilize not only lossy spectral and lossy spatial decorrelation modules but also lossless spectral decorrelation and lossy spatial decorrelation modules and *vice versa*.

Different methods have been proposed for hyperspectral data compression, such as 1) predictive coding, e.g., differential pulse code modulation, CCSDS lossless multispectral and hyperspectral image compression standard; 2) vector quantization (VQ); and 3) transform coding, e.g., Karhunen–Loève transform (KLT), DCT, and DWT. VQ methods can be seen as coding of the hyperspectral image in a cube form, where the spatial and spectral decorrelations are processed in one single stage, while predictive and transform coding methods have been typically used to tackle both spectral and spatial decorrelation [30].

The early hyperspectral image compression studies were based on DPCM. This predictive coding technique predicts a current pixel value using the neighboring pixels and makes use of the difference between the real and predicted values. DPCM can be employed for spatial, spectral, and spectral–spatial decorrelation. Spatial predictive methods have been upgraded to perform interband compression via increasing the size of the neighborhood from 2-D to 3-D. However, according to [31], the direct extension from 2-D to 3-D may not always provide tangible

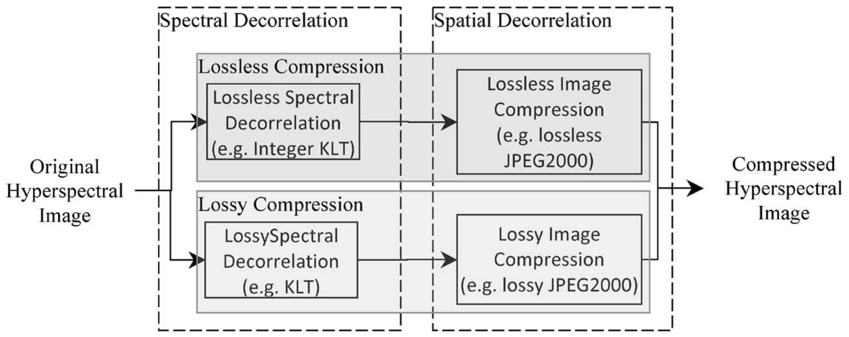

Spectral Decorrelation
Spatial Decorrelation
Lossless Compression
Lossless Spectral Decorrelation (e.g. Integer KLT)
Lossless Image Compression (e.g. lossless JPEG2000)
Original Hyperspectral Image
Lossy Compression
LossySpectral Decorrelation (e.g. KLT)
Lossy Image Compression (e.g. lossy JPEG2000)
Compressed Hyperspectral Image

Fig. 4. *Data flow for hyperspectral image compression.*

benefits, and sometimes can prove to be detrimental. Therefore, it is necessary to develop predictors that are specialized for 3-D hyperspectral images.

Vector quantization is a form of pattern recognition, where an input pattern (i.e., the hyperspectral image) is "approximated" by a predetermined set of standard patterns. The set of standard patterns is also known as a codebook. The difference between the original data and the approximated data and the codebook address are the compressed data that is needed for decompression. Examples that use VQ for hyperspectral image compression are given in [32] and [33]. VQ differs from the compression scheme described in Fig. 4, since it encodes the data in one stage, covering both spatial and spectral domains. However, in [34], DCT was applied in the spectral domain in order to compress the residual data produced by the mean-normalized vector quantization (M-NVQ) algorithm in the spatial domain, which may be seen as compliant with Fig. 4.

In transform coding, the original hyperspectral data are projected by a set of basis vectors to produce a set of product values. The basis vectors differ depending on the particular transform used, e.g., DWT, DCT, or KLT. Transform coding such as DWT and DCT can be used for either spectral or spatial decorrelation in hyperspectral compression. KLT, on the other hand, has been used for spectral decorrelation due to its intrinsic energy-compacting capability that is based on a statistical method. For example, in [35], spectral decorrelation is performed with KLT and spatial decorrelation with DCT; in [36], both spectral and spatial decorrelation are performed with DWT; and in [37], spectral decorrelation is performed with integer KLT and spatial decorrelation with DWT.

At this point, it should be highlighted that CCSDS have recently published a new recommended standard for lossless multispectral and hyperspectral image compression for space applications [38]. It is based on a predictive coding method that depends on the values of nearby samples in the current spectral band (where the reference pixel $s_{z,y,x}$ is located) and P preceding spectral bands, where P is a user-specified parameter. In each spectral band, a local sum of neighboring sample values is calculated through neighbor-oriented or column-oriented approach, which is used to compute one or more local differences within the spectral band. A predicted sample value $\hat{s}_{z,y,x}$ is then calculated by using an adaptive-weighted sum of the local differences in the current and P previous spectral bands. Following that, the difference between $s_{z,y,x}$ and $\hat{s}_{z,y,x}$ is mapped to an unsigned integer $\delta_{z,y,x}$ to produce the predictor output. Finally, the predictor output is coded using an entropy coder that is adaptively adjusted to adapt changes in the statistics of the mapped prediction residuals as defined in [39]. The algorithm provides spatial–spectral decorrelation in one stage.

Finally, it is worth noting that very few remote sensing missions have included hyperspectral image compression capabilities onboard so far. Among the missions that are operational at present only three spacecraft (EO-1, Mars-Express, and IMS-1) perform hyperspectral image compression onboard. However, details about the used compression techniques and their implementation are not available in the open literature. The importance of onboard hyperspectral image compression will grow in the future, as it is expected that newly developed hyperspectral instruments will provide greater spectral coverage, generating tremendous amounts of valuable data [30].

3) FPGA-Based Hyperspectral Image Compression Systems: Due to their critical role in hyperspectral imaging systems, compression algorithms have been mainly implemented on FPGAs for onboard exploitation. For example, in [40], a Xilinx XC3S4000 FPGA is used to accelerate the critical path of a lossless hyperspectral image compression algorithm. A speedup of 21× is achieved when compared with the software version with a processing capability of 16.5 Mpixel/s. The compressor proposed in [41] is also lossless in nature and has been implemented using a Xilinx Virtex-4 LX25 FPGA, resulting in an occupation of 43% and a power consumption of 1.27 watts (W). Compared with the software version the speedup factor was 58× with a maximum data transfer of 58 megasamples per second. In [42], Yu *et al.* have presented a system based on the CCSDS recommendation, capable both of lossless and lossy image compression using three FPGAs from Xilinx: a Spartan3E 12000E-5, a Virtex-4 LX25-12, and a Virtex 5-LX30-3. Results are provided for three different optimization cases: case 1) with no optimizations; case 2) with supreme quantization; and case 3) with supreme quantization and multiplier free. For the best case, a throughput of 250 Mpixels/s at 8 b/pixel was achieved. Depending on the optimizations performed, the power consumption on the Spartan3E was of 309 mW for case 1, 286 mW for case 2, and 273 mW for case 3, demonstrating that optimization efforts have a positive effect in the power consumption. In [43], a new compressor for hyperspectral images is presented. The compressor uses a linear prediction between bands to exploit the interband correlation. Although the authors claim that an FPGA implementation has been developed, only a rough estimation based on other references is provided. Another approach for compression is followed in [44], where Valencia and Plaza use the concept of spectral unmixing to extract the endmembers and abundances and express the remainder pixels as a linear combination of the extracted endmembers. For compression ratios of 20 : 1, 40 : 1, and 80 : 1, the spectral angle similarity scores among the U.S. Geological Survey (USGS) reference signatures from a well-known hyperspectral image collected over the Cuprite mining district in Nevada by the airborne visible infrared imaging spectrometer (AVIRIS)[5] show that the proposed compression system

[5]http://aviris.jpl.nasa.gov

achieves better quality than JPEG2000 multicomponent [22] or set partitioning in hierarchical trees (SPIHT) [45]. The compression algorithm is implemented on a Virtex-II XC2V6000-6 FPGA [46], using only 36% of the resources and, when compared against a sequential implementation using a PC running at 2.6 GHz, the speedup factor was of 70×, using 400 parallel processors and a transfer rate of 40 MB/s. Although the results are obtained very fast (7.94 s for the whole compression procedure), the response is not strictly real-time as the AVIRIS instruments must process a full image cube (614 × 512 pixels with 224 bands) in no more than 5 s to fully achieve real-time performance. This same compression scheme has been also implemented in a GPU from NVidia, the C1060 Tesla GPU, and compared with a FPGA implementation, this time using the Virtex-II PRO xc2vp30 from Xilinx [47]. The total processing time for compressing an AVIRIS hyperspectral scene collected over the World Trade Centre in Manhattan was of 31.23 s using the FPGA and utilizing approximately 76% of the resources at a frequency of 187 MHz. Using the GPU, the total processing time was of 17.59 s. Compared with the sequential processing on an Intel Core i7 920, with a total processing time of 1073.03 s, the speedup factor of the FPGA implementation was of 34.52× and the speedup factor of the GPU was of 61.28×. Even taking into account that the GPU exhibits a speedup factor of 1.77× compared with the FPGA, the power dissipation of this FPGA is of about 221.85 mW [48], while the typical power consumption of this particular GPU is about 200 W. This means that the GPU consumes 901 times more power than the FPGA, achieving only 1.77 times more speed than the FPGA.

B. State-of-the-Art Hyperspectral Linear Unmixing Implementations on FPGAs

The number and variety of information extraction tasks in hyperspectral remote sensing is enormous [49]. However, one of the main problems in hyperspectral data exploitation is the presence of mixed pixels, which arise when the spatial resolution of the sensor is not enough to separate spectrally distinct materials. Furthermore, no matter what the spatial resolution is, the spectral signatures collected in natural environments are invariably a mixture of the signatures of the various materials found within the spatial extent of the ground instantaneous field view of the imaging instrument. Within this context, spectral unmixing has rapidly become one of the most popular techniques to analyze hyperspectral data, which allows for subpixel characterization [50].

1) Basics of Hyperspectral Linear Unmixing: Spectral unmixing generally comprises two stages: 1) identification of pure spectral signatures (endmembers) in the data; and 2) estimation of the abundance of each endmember in each (possibly mixed) pixel [50]. A standard technique for spectral mixture analysis is linear spectral unmixing, which assumes that the collected spectra at the spectrometer can be expressed in the form of a linear combination of endmembers weighted by their corresponding abundances. It should be noted that the linear mixture model assumes that the secondary reflections and the scattering effects can be neglected in the data collection procedure, and, hence, the measured spectra can be expressed as a linear combination of the spectral signatures of materials present in the mixed pixel. If the impact of the secondary reflections or the scattering effects is relevant, more complex nonlinear models can be applied but they demand *a priori* information about the geometry and physical properties of the observed objects and also increase the computational complexity [49]. In the following, we will focus on the linear model since it is the most commonly applied one for hyperspectral unmixing.

To define the linear spectral unmixing problem in mathematical terms, let us assume that a remotely sensed hyperspectral scene with n bands is denoted by I, in which the pixel at the discrete spatial coordinates (i,j) of the scene is represented by a vector $X(i,j) = [x_1(i,j), x_2(i,j), \ldots, x_n(i,j)] \in \mathbb{R}^n$, where $\mathbb{R}$ denotes the set of real numbers in which the pixel's spectral response $x_k(i,j)$ at sensor channels $k = 1,\ldots,n$ is included. Under the linear mixture model assumption, each pixel vector in the original scene can be modeled using

$$x(i,j) \approx \sum_{j=1}^{p} e_j \cdot \boldsymbol{\Phi}_j(i,j) + w = E \cdot \boldsymbol{\Phi} + w \qquad (1)$$

where e_j denotes the spectral response of an endmember, $\boldsymbol{\Phi}_j$ is a scalar value designating the fractional abundance of the endmember e_j, p is the total number of endmembers, and w is a noise vector. It should be noted that $E = \{e_j\}_{j=1}^{j=p}$ can be seen as an $n \times p$ matrix. The solution of the linear spectral mixture problem described in (1) relies on the correct determination of a set of p endmembers and their correspondent abundance fractions $\boldsymbol{\Phi} = \{\boldsymbol{\Phi}_j\}_{j=1}^{j=p}$ at each pixel x.

Over the last decade, several algorithms have been developed for automatic or semiautomatic extraction of spectral endmembers [50], [51]. Winter's N-FINDR [56] has been one of the most successfully applied techniques for automatically determining endmembers in hyperspectral image data. The algorithm attempts to automatically find the simplex of maximum volume that can be inscribed within the hyperspectral data set. The procedure begins with a random initial selection of pixels [see Fig. 5(a)]. Every pixel in the image must be evaluated to refine the estimate of endmembers, looking for the set of pixels that maximizes the volume of the simplex defined by selected endmembers. The corresponding volume is calculated for every pixel in each endmember position by replacing that endmember and finding the resulting volume [see Fig. 5(b)]. If the replacement results in an increase of

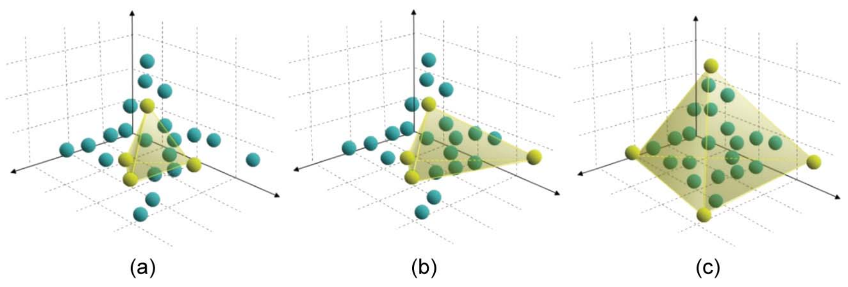


Fig. 5. ***Graphical interpretation of the N-FINDR algorithm in a 3-D space. (a) N-FINDR initialized randomly** ($p = 4$)**. (b) Endmember replacement. (c) Final volume estimation by N-FINDR.***

volume, the pixel replaces the endmember. This procedure is repeated until there are no more endmember replacements [see Fig. 5(c)].

On the other hand, the image space reconstruction algorithm (ISRA) [57] is one of the most popular algorithms for abundance estimation in hyperspectral image data, including expectation–maximization maximum likelihood (EMML) [52], fully constrained least squares unmixing (FCLSU) [53], and nonnegative constrained least squares unmixing (NNLSU) [54]. However, these algorithms are computationally intensive and place a heavy burden on computing systems, and hence, they demand efficient hardware for scenarios under tight time constraints.

2) FPGA-Based Hyperspectral Unmixing Systems: In [55], an implementation of the N-FINDR algorithm using a Virtex-4 XC4VFX60 FPGA from Xilinx was developed. This FPGA model is similar to radiation-hardened FPGAs certified for space operation. The experimental results show that the hardware version of the N-FINDR [56] algorithm can significantly outperform an equivalent software version while being able to provide accurate results in near-real-time. The speedup of this implementation, compared with a software description developed in C language and executed on a PC with AMD Athlon 2.6-GHz processor and 512 MB of RAM, is 37.29× for AVIRIS Cuprite (16 endmembers), 38.10× for a hyperspectral image collected also in the Cuprite mining district by EO-1 Hyperion (21 endmembers), and 37.70× for an AVIRIS image collected over the Jasper Ridge biological preserve in California (19 endmembers). This average speedup factor of 37.63× is quite constant across all the images, even taking into account the differences in the number of endmembers. The second main problem, together with the extraction of suitable endmembers, is the estimation of the fractional abundances of endmembers in each pixel of the hyperspectral scene. An FPGA implementation of the abundances computation task using a parallel ISRA [57] is described in [58]. The FPGA was the same as used in [55], i.e., the Virtex-4 XC4VFX60 FPGA from Xilinx. The system includes a direct memory access (DMA) module and implements a prefetching technique to hide the latency of the input/output (I/O) communications, one of the main bottlenecks found in this kind of applications. In [58], the number of ISRA modules used in parallel is 16, achieving a speedup factor of 10× when processing the AVIRIS Cuprite scene and over 12× when it comes to the two Jasper Ridge AVIRIS scenes. The authors also reach the conclusion that, using FPGAs, the execution time scales linearly with the size of the image. On the other hand, the software implementation increases 3.1 times, which is clearly worse than the hardware implementation. In [59] and [60], an FPGA implementation of the pixel purity index (PPI) algorithm [61] for endmember extraction using the Virtex-II PRO XC2VP30 (por coherencia con el resto) was presented. The proposed hardware system is easily scalable and able to provide accurate results with compact size in near-real-time, which makes the reconfigurable system suitable for onboard hyperspectral data processing. Results show that, against the serial software version which takes 3068 s to process the whole image, the proposed FPGA implementation takes only 31 s, using 10 000 skewers and launching 100 skewers in parallel, achieving a speedup factor of 98.96×. Another FPGA implementation of the PPI algorithm is presented in [62]; in this case, the FPGA needs 62 s to process the same image, hence the speedup factor is smaller (49.48×).

In [63], ISRA has been used in hyperspectral imaging applications to monitor changes in the environment and, specifically, changes in coral reef, mangrove, and sand in coastal areas. In particular, the authors have faced the problem using a hardware/software codesign methodology. The hardware units were implemented on a Xilinx Virtex-II Pro XC2VP30 FPGA and the software was implemented on the Xilinx Microblaze soft processor. As has been observed in all the previous references, the main bottleneck found in this implementation was again data transfer. The only implementation data provided in this paper is that the FPGA was divided in three components: numerator, denominator, and multiplier, where each component works at operating frequencies of 93.75, 84.99, and 113.14 MHz, respectively.

Finally, we would like to highlight two very recent works that deal with the FPGA implementation of two different algorithms for hyperspectral endmember extraction. The first one [64] proposes an architecture for implementing onto a generic FPGA device the so-called real-time fast simplex growing algorithm (RT-FSGA), which is derived from the simplex growing algorithm (SGA) [65], [66], together with the fast computation of simplex volumes uncovered in [67]. One of the main advantages of this architecture comes from the fact that it allows the number of endmembers (p) to vary with image data sets to accommodate various values as opposed to being fixed in the N-FINDR published in [55]. Unfortunately, results about the logic resources occupied by the proposed architecture and/or about its maximum running frequency within a FPGA are not available since the authors have not mapped their architecture onto an FPGA, or at least, they have not disclosed their results. In this sense, only results concerning the speedup of a hypothetical implementation onto a generic FPGA working at 50 MHz with respect to a MATLAB description running onto an Intel Core2 Quad CPU@2.5 GHz and 2-GB memory are reported, resulting in a factor that ranges from 229 to 456, depending on the input image. The second of these works [68] proposes a novel FPGA-based architecture implementing the modified vertex component analysis (MVCA) algorithm [69]. In particular, two versions of the MVCA algorithm which differ on the use of floating point or integer arithmetic for iteratively projecting the hyperspectral cube onto a direction orthogonal to the subspace spanned by the endmembers already computed were mapped onto a XC5VSX95T FPGA from Xilinx, which is quite similar to the new generation of radiation hardened reconfigurable FPGAs from the same company (Virtex-5QV series). With respect to the percentage of the FPGA resources occupied by both versions of the proposed architecture, the authors report that the number of slice registers used in the floating point implementation varies from 27% ($p = 3$) to 80% ($p = 15$) and the number of slice lookup tables (LUTs) varies from 21% ($p = 3$) to 61% ($p = 15$), while for the case of the integer implementation, the number of slice registers used varies from the 27% ($p = 3$) to 71% ($p = 15$), while the number of slice LUTs varies from 21% ($p = 3$) to 51% ($p = 15$). Moreover, all the synthesized integer precision architectures can operate with a frequency up to 268.152 MHz, while the maximum frequency achieved for the synthesized floating point architectures has been 210.438 MHz, being these maximum working frequencies independent of the number of endmembers to be extracted thanks to the design strategy followed by the authors. These results demonstrate that the FPGA implementation of the integer version of the MVCA algorithm shows a better performance in terms of hardware resources and processing speed than its floating point counterpart, both of them being capable of processing hyperspectral images captured by the NASA's AVIRIS sensor in real-time once they are loaded in the internal FPGA memories.

C. Other State-of-the-Art Hyperspectral Imaging Implementations on FPGAs

Several other hyperspectral imaging algorithms have been implemented in FPGAs for improved exploitation. In [70], an implementation of independent component analysis (ICA) [71] is made in order to reduce the dimensionality of hyperspectral images. A parallel ICA has been implemented on a Virtex V1000E running at a frequency of 20.161 MHz, and the board transfers data directly with central processing unit (CPU) on the 64-b memory bus is made at the maximum frequency of 133 MHz. Unfortunately, the authors do not provide execution times, and comparisons with other similar ICA implementations [72], [73] are made based only on maximum synthesized frequencies and the size of the observation data sets.

Another important field of application is hyperspectral image classification. The work in [74] explores design strategies and mappings of classification algorithms for a mix of processing paradigms on an advanced space computing system, featuring MPI-based parallel processing with multiple PowerPC microprocessors, each coupled with kernel acceleration via FPGA and/or AltiVec resources (in-chip vector processing engine). The case-study algorithm adopted for this research uses a linearly constrained minimum variance (LCMV) beam-forming approach, as described in [75], and has a similar computational structure to many other algorithms. The design of key components for hyperspectral imaging systems as the autocorrelation matrix calculation, weight computation, and target detection is discussed, while the hardware/software performance tradeoffs are evaluated. Several parallel-partitioning strategies are considered for extending single-node performance to a clustered architecture consisting of a ten-node PowerPC cluster. Each node contained a 1.4-GHz PowerPC 7455 with AltiVec and 1 GB of SDRAM, connected together with Gigabit Ethernet. Additionally, four nodes were equipped with ADM-XRC-4 FPGA boards from Alpha Data, each containing a Xilinx Virtex-4 SX55 FPGA and four independent 4-MB SRAM modules. Speedup factors compared with the serial simple software version and using AltiVec and 64, 256, and 1024 bands were of 6.47×, 7.31×, and 5.41×, respectively, applying the classification algorithm to three 512×512 images. The highest speedup took place for 256 bands. When using FPGAs, the speedup factors were of 2.27×, 7.48×, and 14.23× for the same amount of bands. It is clear that the use of FPGA is meaningful in the case of high-dimensional images in the spectral domain. When using four processing nodes, the speedup factor using AltiVec is 3.46×, with an average power consumption of 178.75 W, including the CPU and a total energy consumption of 101.5 kJ. For the same amount of nodes, the FPGA accelerated version has a speedup of 32.86×, an average power consumption of 172.51 W, and a total energy consumption of 10.3 kJ. Hence, it is clear how the FPGA-based systems are capable of higher execution speedups with lower power consumptions.

Another important hyperspectral application which requires onboard processing is target detection. In the literature, several efforts have been discussed toward the efficient implementation of target detection algorithms in FPGAs. For instance, in [11], several examples of FPGA implementations for target detection algorithms are discussed. Specifically, chapter 15 in [11] generally discusses the use of FPGAs in detection applications and provides specific application case studies in cloud detection and dimensionality reduction for hyperspectral imaging. Chapter 16 in [11] describes an FPGA implementation of constrained energy minimization (CEM), which has been widely used for hyperspectral target detection applications [76]. The implementation is carried out using the coordinate rotation digital computer (CORDIC) algorithm [76] to convert a Givens rotation of a vector to a set of shift–add operations. This strategy allows for efficient implementation of the CEM in FPGA architectures. Chapter 17 in [11] describes an onboard real-time processing technique for fast and accurate target detection and discrimination in hyperspectral data. In [77], a target detection system based on constraint linear discriminant analysis (CLDA) [78] is implemented using FPGAs. The system is aimed at real-time or near-real-time target detection using hyperspectral data. Although the paper only provides resources consumed by the FPGA and total cycles for the matrix inversion, the authors claim that the developed hardware accelerator is able to achieve real-time or near-real-time throughput depending on the input pixel vector size. Recently, Bernabe *et al.* [79] presented an FPGA design for efficient implementation of the automatic target detection and classification algorithm (ATDCA) [76] on two different kinds of FPGA architectures: Xilinx Virtex-5 and Altera Stratix-III. The experimental results indicate significant performance gains in parallel execution of the ATDCA algorithm but not with real-time performance. In [80], a real-time target detection architecture for hyperspectral image processing is presented and discussed. The proposed architecture is designed and implemented in FPGA to illustrate the relationship between hardware complexity and execution throughput of hyperspectral image processing for target and anomaly detection [76], obtaining relevant results in terms of real-time performance in the context of different applications. In [81], real-time implementations of several popular detection and classification algorithms for hyperspectral imagery are discussed. The adopted approach to real-time implementation of such algorithms is based on using a small portion of hyperspectral pixel vectors in the evaluation of data statistics. An empirical rule of an appropriate percentage of pixels to be used is investigated, which results in reduced computational complexity and simplified hardware implementation. Last but not least, Baker *et al.* [82] recently discussed the problem of implementing matched filters on FPGAs, the Cell IBM processor, and GPUs. The matched filter is an important technique for the processing of hyperspectral data, particularly in the context of target detection applications. In this work, the performance of a matched filter algorithm implementation on an FPGA-accelerated coprocessor (Cray XD-1), the IBM Cell microprocessor, and the NVIDIA GeForce 7900 GTX GPU graphics card has been evaluated. The authors implement only a reduced set of operations for the matched filter, although it is fair to mention that the most time-consuming parts have been the ones considered. In all the cases, the CPU, bus interfaces, power supplies, and involved support hardware are included in all of the performance measurements, including power. The speedup factor (compared with the serial CPU version) of the FPGA was of 3.91× with a power dissipation of 350 W. The speedup factor of the GPU version was of 3.1×, with a power dissipation of 350 W. Finally, the speedup factor of the Cell processor was of 8×, with a power dissipation of 315 W. In this case, the best performance, considering speedup and power, is achieved by the Cell IBM processor, mainly due to its native vector processing arithmetic, which easily maps the matched filter modules.

IV. FIRST CASE STUDY: COMPRESSION OF HYPERSPECTRAL IMAGES

In this section, we present an image compression system based on the KLT as a case study. In order to provide a better understanding, the contents of this section have been divided into three subsections. Section IV-A introduces the KLT and its reversible version, known as integer KLT. Section IV-B details the structure of the spectral decorrelation modules based on the KLT and integer KLT transforms proposed in this work, and, finally, Section IV-C discloses the most significant results achieved.

A. The KLT and Integer KLT Transforms

Spectral decorrelation is a unique component in hyperspectral compression systems, while spatial decorrelation is similar to 2-D image compression performed in other applications. This section describes the results of a research effort, aimed at the development of lossless and lossy spectral decorrelators for satellite hypesrpectral image compression using KLT. The work enables the onboard implementation of all hyperspectral image compression options, shown in Fig. 4, assuming that existing spatial decorrelation solutions are employed, such as the one reported in [42].

The KLT is an orthogonal linear transform, which is applied to 3-D data sets to decorrelate data across different bands, leading to a more compressible data set. KLT, which is strongly related to the well-known principal component analysis (PCA) technique, has been used for multicomponent image compression, not only in remote sensing applications but also in magnetic resonance imaging (MRI), facial recognition, as a technique for finding patterns in high-dimensional data, etc. KLT is considered the optimum method to spectrally decorrelate multispectral data with a number of spectral bands above four according to [35]. The

Table 1 KLT Computational Requirements

Process \ Operations	Additions	Subtractions	Multiplications	Divisions	Trigonometric operations
BandMean	$M{\times}L{\times}n$	-	-	n	-
MeanSub	-	$M{\times}L{\times}n$	-	-	-
Covariance Matrix	$0.5{\times}M{\times}L{\times}n$ $\times(n+1)$ $+M{\times}L{\times}n$	n^2	$0.5{\times}M{\times}L{\times}n{\times}(n+1)$ $+0.5{\times}n(n+1)+n^2$	n^2+n	-
Eigenvectors	$5(n-1){\times}(n-2)$ $\times(2n+4)$	$5(n-1)$ $\times(n-2)$ $\times(2n+4)$	$10(n-1)(n-2)$ $\times(2n+7)$	$10(n-1)$ $\times(n-2)$	$15(n-1)$ $\times(n-2)$
Eigenvectors × MeanSub	$M{\times}L{\times}n$ $\times(n-1)$	-	$M{\times}L{\times}n^2$	-	-
Overall	$1.5{\times}M{\times}L{\times}n$ $\times(n+1)$ $+5(n-1){\times}(n-2)$ $\times(2n+4)$	$M{\times}L{\times}n$ $+5(n-1)$ $\times(n-2)$ $\times(2n+4)$ $+n^2$	$0.5{\times}M{\times}L{\times}n$ $\times(3n+1)$ $+0.5n(n+1)$ $+10(n-1)$ $\times(n-2)(4n+7)$ $+n^2$	$10(n-1)$ $\times(n-2)$ $+n(n+2)$	$15(n-1)$ $\times(n-2)$

KLT algorithm was reported as the best spectral decorrelator for hyperspectral image compression in [83] where JPEG2000 was used as the spatial compression method.

The reversible version of KLT, which is called the integer KLT, performs lossless spectral decorrelation. Experimental results have confirmed that the integer KLT outperforms other techniques in terms of compression ratio when applied to lossless hyperspectral compression [84].

However, the highly computationally intensive nature of KLT and integer KLT is a major challenge when implementing these algorithms. In addition, KLT does not have a fast computation scheme, unlike other transforms, such as DCT and the discrete Fourier transform (DFT). The KLT computation flow consists of the following processes [85]:

- BandMean—obtains the mean of each band;
- MeanSub—subtracts each band from its mean;
- covariance matrix—obtains the covariance matrix of the MeanSub;
- eigenvectors—obtains the eigenvectors of the covariance matrix, which represent the principle components of the image;
- multiplication of the eigenvectors by the MeanSub.

While some of the computation processes above are simple repetitive operations, such as BandMean and MeanSub, others, such as the covariance matrix and the eigenvector evaluations, are more complicated processes involving various sequential operations. Table 1 summarizes the KLT computational requirements, where M and L represent the spatial coordinates and n is the number of bands of the hyperspectral image, represented as a 3-D array of $M \times L \times n$. From Table 1, it can be seen that the computational intensity is proportional to the dimensions of the image. It can also be concluded that the number of the multiplication, addition, and subtraction operations is significantly higher than that of the division and trigonometric operations [85].

A distinctive feature of the KLT algorithm, when it is employed in hyperspectral image compression, is that the size of the processed matrix in the eigenvectors computations is very high, since its dimensions are equal to the number of the spectral bands, which are usually in the order of 100 s. The resultant computational complexity can be overcome through a clustering approach to KLT, which is found to reduce the processing time and memory requirements although it compromises the compression ratio [86]. The negative effect on the compression ratio can be minimized by increasing the size of the clusters. In addition, it is found that selecting the cluster size to be equal to a power of 2 (i.e., 2, 4, 8, 16, 32, etc.) speeds up the multiplications in the KLT covariance and BandMean modules [87].

The integer KLT, proposed in [88], which represents the output in an integer form, is an approximation of KLT, based on matrix factorization. Similarly to KLT, the processes involved in the integer KLT computation are performed on large matrices and, therefore, they are computationally intensive, which slows down the integer KLT evaluation significantly. In addition to BandMean, covariance matrix, and eigenvectors computations as in the original KLT, the integer KLT includes two more complex sequential processes: matrix factorizations and lifting. As in KLT, the compression of a hyperspectral image with n number of bands will involve generating an eigenvector matrix A of size $n \times n$ from the covariance matrix between each pair of bands. Matrix factorization will be applied on the A matrix, which is a nonsingular matrix into four $n \times n$ matrices: a permutation matrix P and three other matrices called triangular elementary reversible matrices (TERMs): L (lower TERM), U (upper TERM), and S (lower TERM). The factorization is not unique and depends on the pivoting method used that will affect the error between the integer approximation and the original KLT transformation. The intrinsic energy-compacting capability of KLT will be affected by the factorization, so the error should be minimized as much as possible.

B. Hardware Designs

Although hardware KLT implementations have been proposed previously, very few authors have targeted embedded computing platforms, such as [89], where a parallel approach to the KLT implementation was presented. However, in [89], only hyperspectral images with a limited number of spectral bands (up to eight) were considered. This section discusses novel lossy and lossless spectral decorrelation modules for a hyperspectral image compression system onboard satellites based on the KLT and integer KLT transforms, respectively.

The proposed designs are targeted at the SmartFusion system-on-a-chip (SoC) platform, which incorporates a Flash-based FPGA fabric and a 32-b hardwired microcontroller. This heterogeneous SoC embedded platform supports a software–hardware codesign approach, which allows tackling the extreme computational complexity of the algorithms via splitting the functionality between a dedicated hardware accelerator and a powerful RISC microprocessor. In particular, the A2F200 SmartFusion SoC has been used in this work, which includes a 32-b ARM Cortex M-3 microcontroller subsystem (MSS). The FPGA logic, which is based on the ProASIC3 device, runs at 50 MHz, while the Cortex M-3 runs at 100 MHz.

The KLT algorithm has been mapped onto the SmartFusionSoC dividing the constituent computational processes between the embedded Cortex M-3 processor and the hardware accelerator on the FPGA fabric using two different approaches as detailed in [87]. A hardware accelerator (coprocessor) is built within the FPGA fabric. The frequently occurring operations are performed in the FPGA fabric to accelerate the execution; while the less frequently occurring ones, such as high-level management and task scheduling, are executed by the embedded Cortex M-3 processor. The BandMean process requires only sequential addition and division operations on a very large set of data and, if implemented on the hardware coprocessor, will lead to an intensive exchange of data between the Cortex M-3 and the FPGA fabric, which will consume a significant time. Therefore, they cannot be efficiently implemented on the hardware coprocessor. The same applies to the MeanSub process, where only subtraction operations are involved. In the first of the two aforementioned approaches, the coprocessor executes the covariance matrix calculation, the most computationally intensive parts of the eigenvectors calculation and the multiplication eigenvectors $\times$ SubMean. In the second approach, only the covariance matrix and the matrix multiplication eigenvectors $\times$ SubMean are executed within the hardware coprocessor. The rationale behind that is to reduce the bit width of the data path. In the first approach, the hardware accelerator utilizes a 32-b data width. However, while the eigenvectors calculation requires 32-b operations, both the covariance and the matrix multiplication (eigen $\times$ SubMean) are performed on a 12-b-wide data path. Therefore, excluding the eigenvectors computations from the hardware coprocessor makes all the operations performed on a 12-b-wide hardware data path, freeing hardware resources.

The integer KLT algorithm has been mapped onto the SmartFusionSoC following a similar software–hardware codesign strategy as with the KLT design above. It is found that the computations of the covariance matrix and lifting scheme take the most of the execution time of the integer KLT. Therefore, by implementing these two processes into the FPGA, a significant acceleration can be achieved. Details of the design can be found in [90].

The KLT and integer KLT designs, discussed above, lend themselves very well to a combined hardware implementation, in which the designs share some of the computational modules. Such a unified design, performing both functions, would take less hardware resources than the two individual ones. This is possible because the KLT and integer KLT algorithms are never used simultaneously, due to the different types of the interband compression that they realize.

A possible mapping with regards to the SmartFusion SoC is to execute the computations of the covariance matrix and all the matrix multiplications in the FPGA fabric, while the rest of the operations are performed by the Cortex M-3 processor. According to a preliminary estimate of the hardware resources, such a combined joint design will only require 10% more system gates and 25% more SRAM blocks than the individual designs.

C. Implementation Results

The AVIRIS hyperspectral images are used as test images in the KLT implementation process. In particular, a portion of the AVIRIS Cuprite image,[6] composed of 512 $\times$ 512 pixels, was employed in the experimental work, presented in this section. The KLT design uses a signed 32-b fixed-point data format, where the first bit represents the sign, the next 16 b represent the integer number, and the remaining 15 b represent the fractional part. An error will be accumulated during the processing, the size of which depends on the matrix size and range of the input data. The effect of using this data format on the compression performance compared with a floating point format is an average compression ratio reduction of less than 5% [87].

In order to assess the performance of the proposed system in terms of execution time, the KLT algorithm was implemented on a PC with an Intel Dual core (2.14 GHz)

[6]http://aviris.jpl.nasa.gov/freedata

Table 2 Execution Times (in Seconds) of the KLT Design for a Cluster of 32 Bands (AVIRIS Cuprite Scene)

	PC	Cortex	App 1	Improvement %	App2	Improvement %
BandMean	0.22	2.94	2.94	-	2.94	-
MeanSub	0.90	4.12	4.12	-	4.12	-
Covariance	13.80	211.65	144.74	31.6%	113.60	46.3%
Eigenvector	3.50	3.62	1.71	52.9%	3.62	-
Eigen × SubMean	16.80	445.82	292.74	34.3%	181.22	59.4%
Overall KLT	35.20	668.15	446.25	33.2%	305.40	54.3%

processor and on the embedded Cortex M-3 processor. The latter was compared with the latency of the proposed SoC system, operating at 100 MHz. The execution times of each process of the KLT algorithm are presented in Table 2 [85]. The latencies of the PC and Cortex M-3 implementations illustrate the computation intensity of each process, confirming that the matrix multiplication (eigen × SubMean) and the covariance matrix computation are the most computationally intensive operations, which together consume more than 95% of the overall processing time.

As can be seen in Table 2, the iterative nature of the eigenvectors computations makes the hardware acceleration very efficient in the first approach (App1), outlined above. In fact, the execution time is reduced by more than 50% while efficient accelerations of 31.6% and 34.3% are achieved on the covariance and the matrix multiplication (eigen × SubMean) processes, respectively. In total, the first approach offers a higher execution speed for the overall KLT algorithm by more than 33% when mapped onto the targeted FPGA-based SoC.

The higher level of parallelism offered by the second approach (App2) leads to a noticeable further improvement in the performance. As can be noticed from Table 2, an acceleration of 46.3% in the covariance process and 59.4% in the matrix multiplication (eigen × SubMean) is achieved, leading to an overall acceleration of 54.3%, cutting the processing time by more than half.

The power consumption of both approaches, estimated using the Actel Smart Power tool, is outlined in Table 3, which shows that it is less than 0.25 W, with the first design approach being less power hungry. For the purpose of comparison, the power consumption of the SoC design introduced in [89] was estimated, and it was found to be more than 2 W [85], which is in stark contrast to the results in Table 3. It can be concluded that the power consumption, offered by the proposed system, is significantly lower despite the four times higher number of processed spectral bands. Table 4 summarizes the required hardware resources for the implementation of the KLT coprocessor using both proposed approaches (including the AMBA Bus interface). Since Flash FPGA devices usualy offer much smaller hardware resources compared to SRAM FPGAs, in both approaches, the designs fitted in the FPGA fabric utilizing less than the full amount of gates.

Table 5 [90] shows the acceleration offered by the proposed integer KLT design. As can be seen from this table, the proposed SoC design accomplishes an acceleration of 44.5% compared to the embedded Cortex M3 processor, almost halving the processing time. The overall power consumption is estimated to be less than 0.25 W. The hardware resources required by the integer KLT design are identical with the resources incurred by the KLT design based on the second approach, as shown in Table 4.

In conclusion, the experimental results presented in this section show that the developed FPGA-based KLT and integer KLT implementations could be used for spectral decorrelation in hyperspectral compression systems onboard remote sensing satellites for certain operational scenarios. However, further research efforts are necessary on reducing the order of complexity of some crucial computations to enable real-time execution of the algorithms.

Table 3 Power Consumption of the KLT Implementations (AVIRIS Cuprite Scene)

	Static	Dynamic	Total
Approach 1	8.99 mW	204.59 mW	213.58 mW
Approach 2	8.99 mW	215.4 mW	224.39 mW

Table 4 Hardware Resources Required by the KLT Implementations (AVIRIS Cuprite Scene)

		Used	Total	Percentage
Approach 1	FPGA Fabric (System Gates)	4248	4608	92%
	Embedded SRAM (Blocks)	4	8	50%
Approach 2	FPGA Fabric (System Gates)	3780	4608	82%
	Embedded SRAM (Blocks)	4	8	50%

Table 5 Execution Time (in Seconds) of the Integer KLT Implementations (AVIRIS Cuprite Scene)

Operations	*Cortex CPU*	*SoC*	*Improvement %*
BandMean	2.943	2.943	-
MeanSub	4.117	4.117	-
Covariance	211.65	113.6	46.3%
Eigenvector	3.625	3.625	-
PLUS	3.93	3.93	-
Lifting	562	309 *	55%
Overall Integer KLT	787.6	437.2	44.5%

*preliminary results

V. SECOND CASE STUDY: SPECTRAL UNMIXING OF HYPERSPECTRAL IMAGES

In this section, we present spectral unmixing as a hyperspectral image processing case study. More concretely, the implemented unmixing chain is graphically illustrated by a flowchart in Fig. 6 and consists of two main steps: 1) endmember extraction, implemented in this work using the N-FINDR algorithm [56]; and 2) nonnegative abundance estimation, implemented in this work using ISRA, a technique for solving linear inverse problems with positive constraints [57]. It is worth mentioning that we have selected the ISRA algorithm because it provides the following important features: 1) its iterative unmixing nature allows controlling the quality of the obtained solutions depending on the number of iterations executed; 2) it guarantees convergence after a certain number of iterations; and 3) it provides positive values in the results of the abundances, which is an important consideration in unmixing applications, as the derivation of negative abundances (which is possible if an unconstrained model for abundance estimation is applied) is not physically meaningful.

This section is organized as follows. Section V-A describes in detail the N-FINDR and the ISRA used for endmember extraction and abundance estimation purposes, respectively, while Section V-B introduces a dynamically reconfigurable FPGA implementation of the considered chain and discloses the most significant results achieved.

A. N-FINDR and the ISRA Algorithms

1) The N-FINDR Endmember Extraction Algorithm: In the following, we provide a detailed step-by-step algorithmic description of the original N-FINDR algorithm developed by Winter. It is interesting to note that the algorithm below represents our own effort to delineate the steps implemented by N-FINDR using available references in the literature [56], [91]. However, it is also worth noting that the N-FINDR algorithm has never been fully disclosed. As a result, this description was developed based on the limited published results available and our own interpretation. Nevertheless, the algorithm below has been verified using the N-FINDR software, provided by the authors, where we have experimentally tested that the software produces essentially the same results as the code below, provided that initial endmembers are generated randomly. The original N-FINDR algorithm can be summarized by the following steps.

1) Feature reduction. Since in hyperspectral data typically the number of spectral bands (n) is much

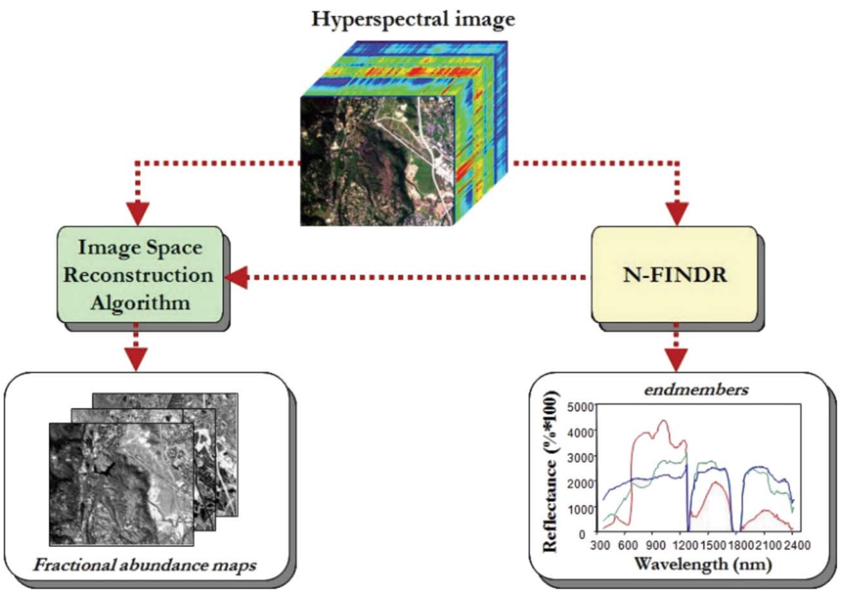


Fig. 6. *Hyperspectral unmixing chain.*

larger than the number of endmembers (p), i.e., $n \gg p$, a transformation that reduces the dimensionality of the input data is required. Hence, the first step consist of applying a dimensionality reduction transformation such as the minimum noise fraction [92] or PCA [93] to reduce the dimensionality of the data from n to $p-1$, where p is an input parameter to the algorithm (number of endmembers to be extracted).

2) Initialization. Let $\{E_1^{(0)}, E_2^{(0)}, \ldots, E_p^{(0)}\}$ be a set of endmembers randomly extracted from the input data.

3) Volume calculation. At iteration $k \geq 0$, calculate the volume defined by the current set of endmembers as follows:

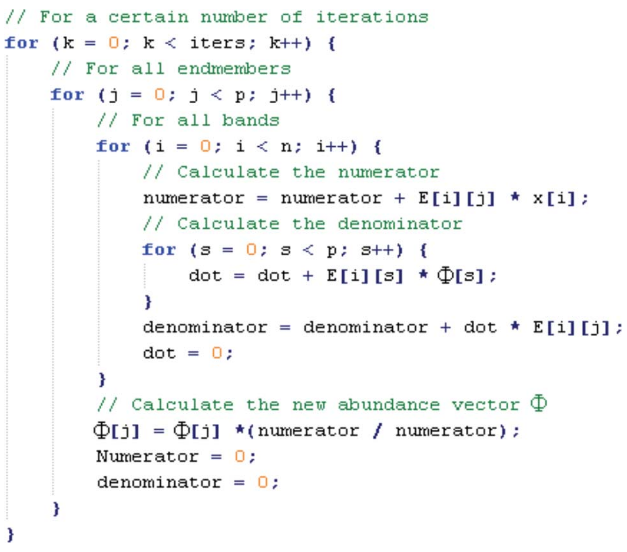
$$V\left(E_1^{(k)}, E_2^{(k)}, \ldots, E_p^{(k)}\right) = \frac{\left|\det\begin{bmatrix} 1 & 1 & \cdots & 1 \\ E_1^{(k)} & E_2^{(k)} & \cdots & E_p^{(k)} \end{bmatrix}\right|}{(p-1)!}. \quad (2)$$

4) Replacement. For each pixel vector $X(i,j)$ in the input hyperspectral data, recalculate the volume by testing the pixel in all p endmember positions, i.e., first calculate $V(X(i,j), E_2(k), \ldots, E_p(k))$, then $V(E_1(k), X(i,j), \ldots, E_p(k))$, and so on, until $V(E_1(k), E_2(k), \ldots, X(i,j))$. If none of the p recalculated volumes is greater than $V(E_1(k), E_2(k), \ldots, E_p(k))$, then no endmember is replaced. Otherwise, the combination with maximum volume is retained. Let us assume that the endmember absent in the combination resulting in the maximum volume is denoted by $E_j^{(k+1)}$. In this case, a new set of endmembers is produced by letting $E_j^{(k+1)} = X(i,j)$ and $E_i^{(k+1)} = E_i^{(k)}$ for $i \neq j$. The replacement step is repeated in an iterative fashion, using as much iterations as needed until there are no more replacements of endmembers.

2) The ISRA Abundance Maps Estimation Algorithm: Once a set of p endmembers $E = \{e_j\}_{j=1}^{j=p}$ has been identified, a positively constrained abundance estimation, i.e., $\Phi_j \geq 0$, can be obtained using ISRA, a multiplicative algorithm based on the following iterative expression:

$$\hat{\boldsymbol{\Phi}}^{k+1} = \hat{\boldsymbol{\Phi}}^{k}\left(\frac{E^T \cdot x}{E^T E \cdot \hat{\boldsymbol{\Phi}}^{k}}\right) \quad (3)$$

where the endmember abundances at pixel x are iteratively estimated, so that the abundances at the $k+1$th iteration $\boldsymbol{\Phi}^{k+1}$ depend on the abundances estimated at the kth iteration $\boldsymbol{\Phi}^{k}$. The procedure starts with an unconstrained



```
// For a certain number of iterations
for (k = 0; k < iters; k++) {
    // For all endmembers
    for (j = 0; j < p; j++) {
        // For all bands
        for (i = 0; i < n; i++) {
            // Calculate the numerator
            numerator = numerator + E[i][j] * x[i];
            // Calculate the denominator
            for (s = 0; s < p; s++) {
                dot = dot + E[i][s] * Φ[s];
            }
            denominator = denominator + dot * E[i][j];
            dot = 0;
        }
        // Calculate the new abundance vector Φ
        Φ[j] = Φ[j] *(numerator / numerator);
        Numerator = 0;
        denominator = 0;
    }
}
```

Fig. 7. ***Pseudocode of ISRA algorithm for unmixing one hyperspectral pixel vector*** x ***using a set of*** E ***endmembers.***

abundance estimation which is progressively refined in a given number of iterations. For illustrative purposes, Fig. 7 shows the ISRA pseudocode for unmixing one hyperspectral pixel vector x using a set of E endmembers. For simplicity, in the pseudocode, x is treated as an n-dimensional vector, and E is treated as an $n \times p$-dimensional matrix. The estimated abundance vector $\boldsymbol{\Phi}$ is a p-dimensional vector, and variable *iters* denotes the number of iterations per pixel in the abundance estimation process. The pseudocode is subdivided into the numerator and denominator calculations in (3). When these terms are obtained, they are divided and multiplied by the previous abundance vector. It is important to emphasize that the calculations of the fractional abundances for each pixel are independent, so they can be calculated simultaneously without data dependencies, thus increasing the possibility of parallelization.

B. FPGA-Based Linear Unmixing of Hyperspectral Images

With reconfigurable hardware, it is possible to apply much of the flexibility that was formally restricted to software developments only. The idea is that FPGAs can be reconfigured on the fly. This approach is called temporal partitioning [94], [95] or runtime reconfiguration [96]. Basically, the FPGA (or a region of the FPGA) executes a series of tasks one after another by reconfiguring itself between tasks [97]. The reconfiguration process updates the functionality implemented in the FPGA, and a new task can then be executed. This time-multiplexing approach allows for the reduction of hardware components onboard since one single reconfigurable module can substitute several hardware peripherals carrying out different functions during different phases of the mission.

To implement the spectral unmixing chain on FPGAs, we will take advantage of their reconfigurability. The idea is to start with the implementation of the N-FINDR algorithm occupying the entire FPGA in order to maximize the parallelization of the endmember extraction algorithm. Once the endmembers are found, we will apply dynamic reconfiguration to replace the N-FINDR algorithm with the parallelized ISRA, occupying again the entire FPGA. In our particular case, the N-FINDR and ISRA algorithms are easily scalable because their basic processing units require few hardware resources. Hence, it will be shown that the FPGA is almost full in both cases. More details about our implementations of the N-FINDR and ISRA modules can be found in [55] and [58], although this is the first time that we have used these two modules to build a spectral unmixing chain using runtime reconfiguration. To this end, we have included support for intertask communication following a shared memory scheme. Thus, the N-FINDR module will store its results in a known memory address that corresponds to an external DDR2 SRAM memory. Once the system has carried out the reconfiguration, the ISRA module will load the data from that memory. Using external memories to store and load the data may introduce significant delays in the execution. As we will explain in detail, we have overcome this problem including a direct memory access module (DMA) in the system and overlapping the memory transfers with useful computations. With this approach, both reconfiguration and communications introduce a negligible overhead.

1) N-FINDR Hardware Design: Fig. 8 describes the general architecture of the hardware used to implement the N-FINDR algorithm, along with the I/O communications. For data input, we use a DDR2 SDRAM and a DMA (controlled by a PowerPC using a prefetching approach) with a first-in–first-out (FIFO) to store pixel data. For data output, we use again the PowerPC to write the position of the

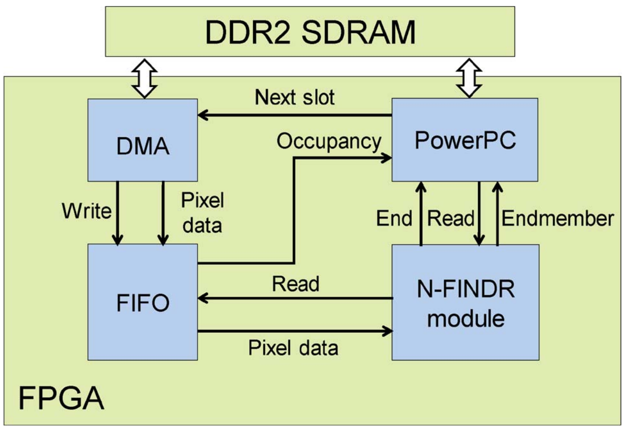


Fig. 8. ***Hardware architecture to implement the endmember extraction step.***

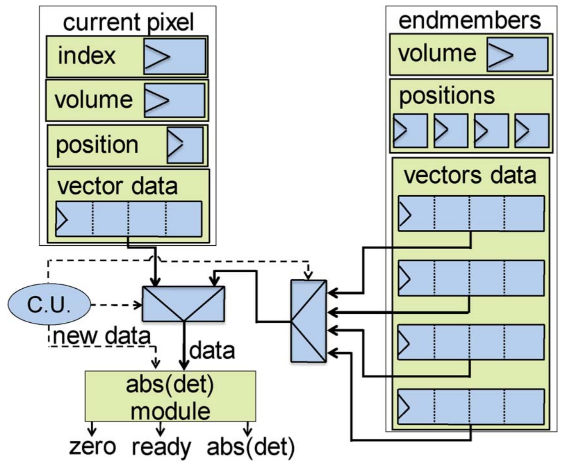


Fig. 9. ***Hardware architecture to implement the N-FINDR algorithm.***

endmembers in the DDR2 SDRAM. Finally, the N-FINDR module is used to implement our version of the N-FINDR algorithm. At this point, it is worth to mention that currently we have not developed a hardware implementation for the PCA or the MNF algorithm in order to carry out the dimensionality step. Hence, we will assume that this step has been previously performed and our hardware implementation will carry out all the remaining steps.

The most time-consuming part of the algorithm is the volume calculation. The limited available resources in a small or medium FPGA to calculate determinants of large order make it difficult to develop an efficient implementation of the algorithm. In our implementation, we have solved this issue by taking advantage of the fundamental properties of the determinants and applying them systematically to transform the determinant in others who are increasingly easy to calculate, down to one that is trivial. For the design of the algorithm we use the matrix triangulation method.

Fig. 9 shows the hardware architecture used to implement the volume calculation step. We use registers to store the pixel vectors selected as endmembers until the current moment, their positions in the image and their volume, and also the current pixel vector data, its position, its greater volume, and the index inside the matrix where it is obtained. Moreover, we have included a module that calculates the absolute value of the determinant using the matrix triangulation process: first, for $j = 2, \ldots, n$, we take a multiple a_{j1}/a_{11} of the first row and subtract it to the jth row, to make $a_{j1} = 0$. Thus, we remove all elements of matrix A below the "pivot" element a_{11} in the first column. Now, for $j = 3, \ldots, n$, we take a multiple a_{j2}/a_{22} of the second row and subtract it to the jth row. When we have finished this, all subdiagonal elements in the second column are zero, and we are ready to process the third column. Applying this process to columns $i = 1, \ldots, n-1$ completes the matrix triangulation process and matrix A

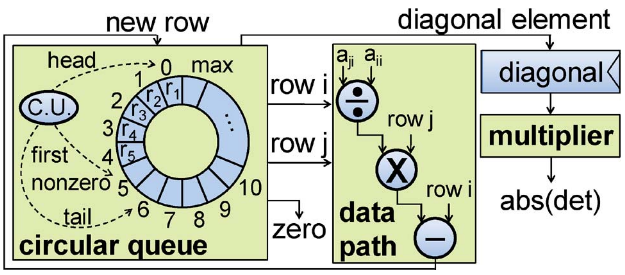


Fig. 10. ***Hardware architecture of the abs(det) module.***

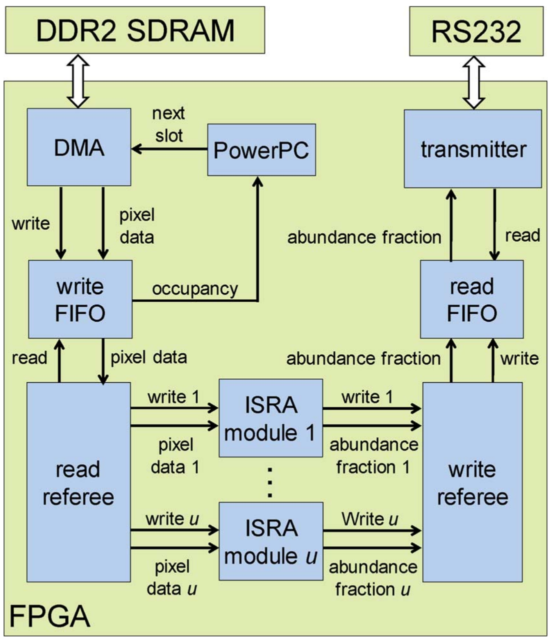


Fig. 11. ***Hardware architecture to implement the fractional abundance estimation step.***

has been reduced to upper triangular form. These operations are carried out by the data path (see Fig. 10).

Obviously, if one of the diagonal pivots a_{ii} is zero, we cannot use a_{ii} to remove the elements below it; we cannot change a_{ji} by subtracting any multiple of $a_{ii} = 0$ from it. We must switch row i with another row k below it, which contains a nonzero element a_{ki} in the ith column. Now the new pivot a_{ii} is not zero, and we can continue the matrix triangulation process. If $a_{ki} = 0$ for $k = i, \ldots, n$, then it will not be satisfactory to switch row i with any rows below it, as all the potential pivots are zero and, therefore, $\det(A) = 0$. This behavior has been implemented using a modified circular queue with a small control unit (see Fig. 10). Finally, the segmented multiplier calculates the multiplication of the main diagonal elements of the triangular matrix and obtains the absolute value. Table 6 summarizes the computational requirements for the N-FINDR hardware implementation, where M and L represent the spatial coordinates, n is the number of bands of the hyperspectral image, and p is the number of endmembers to be extracted.

2) ISRA Hardware Design: Fig. 11 describes the general architecture of the hardware used to implement the ISRA, along with the I/O communications, following a similar scheme to the one in the previous subsection. For data input, we use a DDR2 SDRAM and a DMA (controlled by a PowerPC using a prefetching approach) with a FIFO to store pixel data. ISRA module is used to implement our parallel version of the ISRA. Finally, a transmitter is used to send the fractional abundances via an RS232 port.

Fig. 12 shows the hardware architecture used to implement the ISRA module. Three different memories are used to store the endmembers, the current pixel, and the fractional abundances for the current pixel, respectively. The ISRA data path represents the hardware architecture used to perform the calculations. The control unit carries out the ISRA execution: it reads the appropriate memory locations for each of the memories and updates the fractional abundances. In addition, we use a combinational circuit based on multiplexers to select the appropriate input data. Once calculated, the system writes the estimated abundances to the read FIFO.

Fig. 12 also describes the architecture of the data path used to implement the ISRA module. The dot-product unit is used for calculating both the numerator and the denominator in (3), allowing a proper reuse of hardware resources. To perform the update of a fractional abundance value, it proceeds as follows: during n cycles (where n is

Table 6 N-FINDR and ISRA Computational Requirements (Single-Precision Floating Point Operations)

Operations / Process	Additions	Subtractions	Multiplications	Divisions
N-FINDR	-	$M*L*p*\sum_{j=1}^{p-1} j$	$M*L*p*\left(\sum_{j=1}^{p-1} j+p\right)$	$M*L*p*\sum_{j=1}^{p-1} j$
ISRA	$M*L*i*(n+p*n)$	-	$M*L*i*(n+(p+1)*n+1)$	$M*L*i$

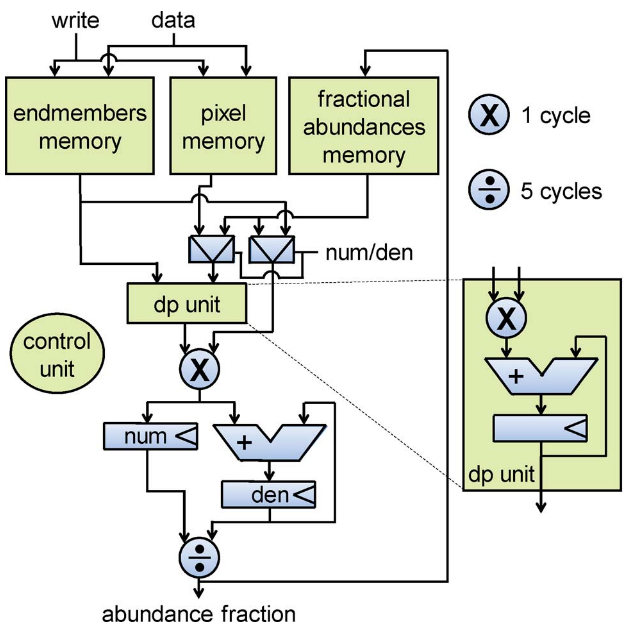


Fig. 12. ***Hardware architecture to implement the ISRA module.***

the number of bands) it computes the dot-product between the current pixel and the endmember corresponding to the proportion of abundance that is being updated. In the next clock cycle, the result of the dot-product is multiplied by the previous abundance fraction and the result is stored in register *num*, thus concluding the calculation of the numerator. To calculate the denominator, the aforementioned procedure is repeated p times (where p is the number of endmembers) with the appropriate input data, while partial results are accumulated using an adder and the register *den*. The calculation of the denominator requires therefore $p \times (n+1)$ clock cycles. The process finalizes with the division between the numerator and the denominator in five clock cycles. The computational requirements of this design are presented in Table 6, where i represents the number of iterations per pixel in the abundance estimation process.

3) *Implementation Results:* The hardware architectures previously described have been implemented on an ML410 board, a reconfigurable board with a single Virtex-4 XC4VFX60 FPGA component, a memory slot which holds up to 2 GB and some additional components not used in our implementation. We have used a Xilinx Virtex-4 XC4VFX60 FPGA because its features are very similar to the space-grade Virtex-4QV XQR4VFX60 FPGA. The results obtained by our FPGA-based unmixing chain were exactly the same as the ones obtained with its equivalent software versions, and the efficiency in terms of unmixing of the N-FINDR and the ISRA algorithms has been already proven in many state-of-the-art works, therefore, we have decided to center this subsection on the hardware implementation results achieved.

Table 7 shows the resources used for our hardware implementation of the proposed N-FINDR algorithm design optimized to extract up to 21 endmembers, and Table 8 shows the resources used for our hardware implementation of the proposed ISRA design for 16 ISRA basic units in parallel, conducted on the Virtex-4 XC4VFX60 FPGA of the ML410 board. This FPGA has a total of 25 280 slices, 50 560 slice flip flops, and 50 560 four input lookup tables available. In addition, the FPGA includes some heterogeneous resources, such as two PowerPCs, 128 DSP48Es, and distributed Block RAMs. Table 9 outlines the power consumption of both designs, estimated using the Xilinx Power Estimator tool. Taking into account that we are using most of the FPGA computing resources simultaneously, these numbers are very affordable, especially when compared to the power numbers of high-performance general-purpose processors or GPUs. Finally, Table 10 reports the processing times measured for two well-known hyperspectral data sets not only for our FPGA implementation, but also for an equivalent software version developed in C language and executed on a PC with an Intel Core i7 processor at 2.2 GHz and 4 GB of RAM. The first data set corresponds with a portion of 350 × 350 pixels of the AVIRIS Cuprite scene also used for the compression case study reported in the previous section, while the second data set corresponds to an EO-1 Hyperion data set available in radiance units collected over the same Cuprite mining district as the aforementioned AVIRIS scene. In this case, we used a full EO-1 Hyperion flightline with much larger dimensions, i.e., 6479 × 256 pixels and

Table 7 Summary of Resource Utilization for the FPGA-Based Implementation of the N-FINDR Algorithm on a VIRTEX-4 XC4VFX60 FPGA

Component	Number of endmembers (p)	Number of DSP48Es	Number of RAMB16s	Number of slice flip flops	Number of 4 input LUTs	Number of slices	Percentage of total	Maximum operation frequency (MHz)
N-FINDR module	21	128	217	20077	34155	24622	97.40	51.3
DMA Controller	-	0	0	170	531	367	1.45	102

Table 8 Summary of Resource Utilization for the FPGA-Based Implementation of the ISRA Algorithm on a VIRTEX-4 XC4VFX60 FPGA

Component	Number of modules (*u*)	Number of DSP48Es	Number of RAMB16s	Number of slice flip flops	Number of 4 input LUTs	Number of slices	Percentage of total	Maximum operation frequency (MHz)
Parallel ISRA module	16	128	54	5532	42047	22773	90.08	53.4
RS232 Transmitter	-	0	0	69	128	71	0.28	208
DMA Controller	-	0	0	170	531	367	1.45	102

242 spectral bands, and a total size of around 800 MB. In both cases, our hardware implementation achieves a speedup of 9×, which is a remarkable result if it is taken into account that the Intel Core i7 uses a much more recent technology (our Virtex-4 uses 2004 technology whereas the Intel Core i7 uses technology from the end of 2011), and is a power hungry platform that demands complex heat dissipation systems, which are not needed for our FPGA design.

To conclude this section, we emphasize that our reported FPGA processing times are still slower than the sensors that collect the hyperspectral data. For instance, the cross-track line scan time in EO-1 Hyperion, a push-broom instrument, is quite fast (4.5 ms to collect 256 full pixel vectors). This introduces the need to process a hyperspectral data set with 256 × 6479 pixels (such as our considered EO-1 Hyperion Cuprite scene) in 29.16 s to fully achieve real-time performance. Although we believe that the inclusion of more recent FPGA boards could significantly improve our reported processing times, we have decided to report results on a board which is very close to those certified for space operation in the hope of illustrating the real challenges to be faced when porting hyperspectral imaging algorithms into certified FPGA hardware. For example, in a Virtex-4 FPGA XQR4VLX200 (89 088 slices) certified for space, we could have 3.5 times more ISRA units in parallel simply synthesizing the second stage of the hardware architecture for the new number of units in parallel. In this way, we could achieve a speedup of 3.5× with regards to our current implementation without any modifications in our proposed design. In the case of an airborne platform without the need for space-certified hardware, we could use a Virtex-6 XQ6VLX550T (550 000 logics cells) which has nearly ten times more logic cells than the FPGA used in our experiments. In future developments, we will also focus on improving our proposed implementation to achieve a better utilization of hardware resources and reduce the reported processing times, which in any event are considered to be acceptable in many remote sensing applications.

Table 9 Power Consumption of the N-FINDR and ISRA Designs

	Static	*Dynamic*	*Total*
N-FINDR	455 mW	350 mW	805 mW
ISRA	455 mW	362 mW	817 mW

VI. SUMMARY AND FUTURE RESEARCH

This paper discussed the role of reconfigurable hardware in remote sensing missions by providing an extensive review and analysis of the (current and future) capabilities of FPGA devices, which are now the standard for onboard processing and analysis of remotely sensed data. Both the technological aspects of FPGA hardware and also specific implementation issues are covered in the context of two specific case studies dealing with remotely sensed hyperspectral data compression and spectral unmixing. While many other techniques currently exist for hyperspectral data exploitation, the selection of these two applications was based on their relevance and fast development in recent years. Several summarizing statements can be formulated after our detailed assessment.

- FPGAs currently represent the best choice for onboard hyperspectral processing due to their compact size and low weight, particularly when compared with other specialized hardware devices such as GPUs, which exhibit much higher power dissipation figures. FPGAs also perform well in high-throughput signal processing tasks encountered in space, like processing and/or compressing a hyperspectral image.
- An important property of FPGAs is their inherent ability to change their functionality—through partial or full reconfiguration. In other words, they permit changes to the usage model and the data processing paradigm in space rather than hard

Table 10 Execution Times Measured for the FPGA Hardware Implementation (HW) and for an Equivalent Software (SW) Version

	AVIRIS Cuprite subscene (16 endmembers and 10 iterations)		EO-1 Hyperion Cuprite scene (21 endmembers and 10 iterations)	
	HW	SW	HW	SW
Endmember extraction (N-FINDR)	13.46 s	476.34 s	239.11 s	8464,99 s
Dynamic reconfiguration	694 ms	0 ms	694 ms	0 ms
Fractional abundance estimation (ISRA)	1.32 min	5.81 min	30.21 min	173.4 min
Total	1.56 min	13.75 min	34.21 min	314.48 min

coding of all components prior to launch. Although FPGAs that support runtime reconfiguration have been available for many years, initially the reconfiguration flow was too complex, and very few designers' teams use it. In fact, the example described in Section V is the first time where runtime reconfiguration has been used for hyperspectral images. However, in recent years, FPGA vendors have put in a great effort to simplify the development of runtime reconfigurable systems. Hence, future onboard processing systems could easily take advantage of this useful feature.

- Another important technological advantage of FPGAs over commodity high-performance platforms such as current GPUs is that FPGAs can be manufactured in order to resist high levels of radiation without changing the content of their inner memories, which determines the FPGA programming. Moreover, since FPGAs implement custom designs, additional fault-tolerant levels can be included in the system when needed, such as dual or triple modular redundancy.
- The aforementioned properties make FPGAs a very attractive solution for advanced hyperspectral data exploitation applications such as (lossless and lossy) hyperspectral data compression and unmixing, as discussed in this work. However, FPGAs cannot still provide real-time processing performance onboard in the considered analysis scenarios. This is mainly due to the additional hardware resources required by the reconfigurability property, and also to the extra resources needed to enable adaptive execution of a particular processing algorithm among a suite of algorithms, implementing a desired functionality. In addition, in view of the computationally intensive nature of the hyperspectral imaging algorithms new low-complexity mapping hardware methods are required enabling real-time execution based on an appropriate tradeoff among design requirements. In this regard, it is important to mention that due to the time needed for the certification process of hardware, current space-certified FPGA devices do not represent the state-of-the-art hardware components, hence the technology available onboard is generally one generation behind the latest one available in the market.
- Another consideration when porting processing algorithms to FPGAs is the programming effort needed, which is generally higher than the one required to develop an efficient implementation in a GPU or multicore processor since the full processing architecture needs to be developed from scratch. High-level design tools such as Handel-C,[7] Catapult C,[8] Dime-C,[9] or Impulse C[10] among others have simplified the design process for FPGA developers, but the learning curve for inexperienced users implementing their algorithms in FPGA technology is still significant as compared to the efforts needed in other kinds of commodity high-performance computing platforms such as GPUs or multicores. In this sense, some efforts have been done in order to directly map hyperspectral imaging algorithms described in MATLAB (which is by far the preferred programming language within the hyperspectral research community) onto a generic FPGA, but the results are still far from being efficient when compared with the classical FPGA design flow [79], [98]. In any case, all the FPGA vendors are trying to reduce the design complexity not only including support to synthesize high-level language such as C or C++, but also including large component intellectual property (IP) libraries with hundreds of frequently used

[7]http://www.handelc.com
[8]http://www.calypto.com/catapult_c_synthesis.php
[9]http://www.nallatech.com/Development-Tools/dime-c.html
[10]http://www.impulseaccelerated.com/

designs, and allowing the easy integration of previously designed hardware blocks in a custom design.

In summary, reconfigurable hardware offers an unprecedented opportunity to achieve the long-desired goal of being able to perform real-time processing of high-dimensional remote sensing data (such as hyperspectral images) onboard an instrument. FPGA technology is now highly consolidated in both airborne and spaceborne scenarios, and provides an important property such as reconfigurability which allows for a longer exploitation of the hardware and its dynamic adaptation to the availability of new processing techniques and algorithms. Because of these reasons, FPGA technology has attracted the attention of international space agencies for their future onboard processing systems. A good example is the SpaceCube program of the NASA Goddard Space Flight Center, which aims to provide 10× to 100× improvements in onboard computing power while lowering relative power consumption and cost thanks to the incorporation of commercially available radiation-tolerant FPGAs to their flight processing systems [99].

In this regard, the review provided in this contribution analyzes current and future developments in this area and further discusses the role of FPGAs in future spaceborne missions for Earth observation, intended to provide full coverage of our planet with extremely high spatial, spectral, and temporal resolutions. Expected developments in this field will comprise the full integration of reconfigurable technology into spaceborne missions and a more dynamic certification process by international agencies that will allow for the integration of state-of-the-art hardware devices into remote sensing missions as soon as they become available. Further developments toward enhancing the programmability of FPGA devices are also expected in order to facilitate the porting of algorithms to this kind of hardware using high-level tools. ■

Acknowledgment

The authors would like to thank the Guest Editors of this special issue for their very kind invitation to provide a contribution, as well as the three anonymous reviewers for their outstanding comments and suggestions for improvement.

REFERENCES

[1] R. Trautner, "ESA's roadmap for next generation payload data processors," in *Proc. DASIA Conf.*, 2011. [Online]. Available: http://www.esa.int/TEC/OBDP/

[2] *Zynq-7000 All Programmable SoC*. [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/index.htm

[3] *Altera SoC FUGA*. [Online]. Available: http://www.altera.com/devices/processor/soc-fpga/proc-soc-fpga.html

[4] S. Hauck and A. Dehon, Eds."Device architecture," in *Reconfigurable Computing*. Amsterdam, The Netherlands: Elsevier, 2008, pp. 3–27.

[5] *Space-Grade Virtex-4QV FUGA*. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-4qv/index.htm

[6] *Space-Grade Virtex-5QV FUGA*. [Online]. Available: http://www.xilinx.com/products/silicon-devices/fpga/virtex-5qv/index.htm

[7] *Reports on Radiation Effects in Xilinx FPGAs*. [Online]. Available: http://parts.jpl.nasa.gov/organization/group-5144/radiation-effects-in-fpgas/xilinx/

[8] *ProASIC3 FPGA*. [Online]. Available: http://www.actel.com/products/pa3/

[9] T. Kuwahara, "FPGA-based reconfigurable on-board computing systems for space applications," Ph.D. dissertation, Faculty Aerosp. Eng. Geodesy, Univ. Stuttgart, Stuttgart, Germany, 2009.

[10] A. Plaza, Q. Du, Y.-L. Chang, and R. L. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 528–544, Sep. 2011.

[11] A. Plaza and C.-I. Chang, *High Performance Computing in Remote Sensing*. Boca Raton, FL: CRC Press, 2007.

[12] A. Plaza, D. Valencia, and J. Plaza, "High-performance computing in remotely sensed hyperspectral imaging: The pixel purity index algorithm as a case study," in *Proc. Int. Parallel Distrib. Process. Symp. (IPDPS)*, 2006.

[13] A. Plaza, J. Plaza, A. Paz, and S. Sanchez, "Parallel hyperspectral image and signal processing," *IEEE Signal Process. Mag.*, vol. 28, no. 3, pp. 119–126, May 2011.

[14] C. A. Lee, S. D. Gasster, A. Plaza, C.-I. Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 508–527, Sep. 2011.

[15] G. Yu, T. Vladimirova, and M. N. Sweeting, "Image compression systems on board satellites," *Acta Astronautica*, vol. 64, no. 9–10, pp. 988–1005, May 2009.

[16] N. S. Jayant and P. Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video*. Englewood Cliffs, NJ: Prentice-Hall, 1984.

[17] M. Rabbani and P. W. Jones, *Digital Image Compression Techniques*. Bellingham, WA: SPIE Press, 1991.

[18] B. V. Brower, D. Couwenhoven, B. Gandhi, and C. Smith, "ADPCM for advanced LANDSAT downlink applications," in *Conf. Rec. 27th Asilomar Conf. Signals Syst. Comput.*, 1993, vol. 2, pp. 1338–1341.

[19] *Lossless Data Compression, Recommendation for Space Data System Standards*, CCSDS 121.0-B-1, May 1997.

[20] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. London, U.K.: Chapman & Hall, 1993.

[21] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *IEEE Trans. Image Process.*, vol. 9, no. 8, pp. 1309–1324, Aug. 2000.

[22] D. S. Taubman and M. W. Marcellin, *JPEG2000: Image Compression Fundamentals, Standard and Practice*. Norwell, MA: Kluwer, 2002.

[23] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Process.*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.

[24] A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 243–250, Jun. 1996.

[25] *Image Data Compression, Recommendation for Space Data System Standards*, CCSDS 122.0-B-1, CCSDS (2005), Nov. 2005.

[26] P.-S. Yeh and J. Venbrux, "A high performance image data compression technique for space applications," in *Proc. NASA Earth Sci. Technol. Conf.*, 2003, pp. 214–228.

[27] J. Serra-Sagrista, C. Fernandez-Cordoba, F. Auli-Llinas, F. Garcia-Vilchez, and J. Minguillon, "Lossy coding techniques for high resolution images," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 5238, pp. 276–287, 2004.

[28] J. Serra-Sagrista, F. Auli-Llinas, F. Garcia-Vilchez, and C. Fernandez-Cordoba, "Review of CCSDS-ILDC and JPEG2000 coding techniques for remote sensing," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 5573, pp. 250–261.

[29] Y. Pen-Shu, P. Armbruster, A. Kiely, B. Masschelein, G. Moury, C. Schaefer, and C. Thiebaut, "The new CCSDS image compression recommendation," in *Proc. IEEE Conf. Aerosp.*, 2005, pp. 4138–4145.

[30] N. R. Mat Noor and T. Vladimirova, "Investigation into lossless hyperspectral image compression for satellite remote sensing," *Int. J. Remote Sens.*, to be published.

[31] G. Motta, F. Rizzo, and J. A. Storer, Eds. "Lossless predictive compression of hyperspectral images," in *Hyperspectral Data Compression*. New York: Springer-Verlag, 2005.

[32] M. J. Ryan and J. F. Arnold, "The lossless compression of AVIRIS images by vector quantization," *IEEE Trans. Geosci. Remote Sens.*, vol. 35, no. 3, pp. 546–550, May 1997.

[33] M. J. Ryan and J. F. Arnold, "Lossy compression of hyperspectral data using

vector quantization," *Remote Sens. Environ.*, vol. 61, no. 3, pp. 419–436, Sep. 1997.

[34] G. Motta, F. Rizzo, and J. A. Storer, Eds. "An architecture for the compression of hyperspectral imagery," in *Hyperspectral Data Compression.* New York: Springer-Verlag, 1995.

[35] J. A. Saghri, A. G. Tescher, and J. T. Reagan, "Practical transform coding of multispectral imagery," *IEEE Signal Process. Mag.*, vol. 12, no. 1, pp. 32–43, Jan. 1995.

[36] G. Liu and F. Zhao, "Efficient compression algorithm for hyperspectral images based on correlation coefficients adaptive 3D zerotree coding," *IET Image Process.*, vol. 2, no. 2, pp. 72–82, Apr. 2008.

[37] I. Blanes and J. Serra-Sagrista, "Clustered reversible-KLT for progressive lossy-to-lossless 3D image coding," in *Proc. Data Compression Conf.*, 2009, pp. 233–242.

[38] *Lossless Multispectral & Hyperspectral Image Compression—Recommended Standard*, CCSDS 123.0-B-1, May 2012.

[39] *Lossless Data Compression—Recommended Standard*, CCSDS 121.0-B-2, May 2012.

[40] Y.-T. Hwang, C.-C. Lin, and R.-T. Hung, "Lossless hyperspectral image compression system-based on HW/SW codesign," *IEEE Embedded Syst. Lett.*, vol. 3, no. 1, pp. 20–23, Mar. 2011.

[41] N. Aranki, D. Keymeulen, A. Bakhshi, and M. Klimesh, "Hardware implementation of lossless adaptive and scalable hyperspectral data compression for space," in *Proc. NASA/ESA Conf. Adapt. Hardware Syst.*, 2009, pp. 315–322.

[42] G. Yu, T. Vladimirova, and M. N. Sweeting, "FPGA-based on-board multi/hyperspectral image compression system," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2009, vol. 5, pp. 212–215.

[43] A. C. Miguel, A. R. Askew, A. Chang, S. Hauck, R. E. Ladner, and E. A. Riskin, "Reduced complexity wavelet-based predictive coding of hyperspectral images for FPGA implementation," in *Proc. Data Compression Conf.*, 2004, pp. 469–478.

[44] D. Valencia and A. Plaza, "FPGA-based hyperspectral data compression using spectral unmixing and the pixel purity index algorithm," in *Proc. 6th Int. Conf. Comput. Sci.*, 2006, pp. 888–891.

[45] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. Circuits Syst.*, vol. 6, no. 3, pp. 243–250, Jun. 1996.

[46] A. Plaza, "Towards real-time compression of hyperspectral images using Virtex-II FPGAs," in *Proc. Int. Euro-Par Conf.*, 2007, pp. 248–257.

[47] A. Plaza, S. Sanchez, A. Paz, and J. Plaza, "GPUs versus FPGAs for onboard compression of hyperspectral data," in *Proc. Int. Workshop On-Board Payload Data Compression*, 2010, pp. 318–324.

[48] S. Deo, "Power consumption calculation of AP-DCD algorithm using FPGA platform," in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2010, pp. 388–393.

[49] A. Plaza, J. A. Benediktsson, J. Boardman, J. Brazile, L. Bruzzone, G. Camps-Valls, J. Chanussot, M. Fauvel, P. Gamba, J. A. Gualtieri, M. Marconcini, J. C. Tilton, and G. Trianni, "Recent advances in techniques for hyperspectral image processing," *Remote Sens. Environ.*, vol. 113, no. supplement 1, pp. 110–122, Sep. 2009.

[50] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical and sparse regression-based approaches," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 5, no. 2, pp. 354–379, Apr. 2012.

[51] A. Plaza, P. Martinez, R. Perez, and J. Plaza, "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Trans. Geosci. Remote Sens.*, vol. 42, no. 3, pp. 650–663, Mar. 2004.

[52] J. M. P. Nascimento and J. M. Bioucas-Dias, "Does independent component analysis play a role in unmixing hyperspectral data?" *IEEE Trans. Geosci. Remote Sens.*, vol. 43, no. 1, pp. 175–187, Jan. 2005.

[53] D. Heinz and C.-I. Chang, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 3, pp. 529–545, Mar. 2001.

[54] M. Velez-Reyes, A. Puetz, M. P. Hoke, R. B. Lockwood, and S. Rosario, "Iterative algorithms for unmixing of hyperspectral imagery," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 5093, pp. 418–429, 2003.

[55] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "FPGA implementation of the N-FINDR algorithm for remotely sensed hyperspectral image analysis," *IEEE Trans. Geosci. Remote Sens.*, vol. 50, no. 2, pp. 374–388, Feb. 2012.

[56] M. E. Winter, "N-FINDR: An algorithm for fast autonomous spectral end-member determination in hyperspectral data," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 3753, pp. 266–275, 1999.

[57] S. Torres-Rosario, "Iterative algorithms for abundance estimation on unmixing of hyperspectral imagery," M.S. thesis, Dept. Electr. Comput. Eng., Univ. Puerto Rico, San Juan, 2004.

[58] C. Gonzalez, J. Resano, A. Plaza, and D. Mozos, "FPGA implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 5, no. 1, pp. 248–261, Feb. 2012.

[59] C. Gonzalez, D. Mozos, J. Resano, and A. Plaza, "FPGA for computing the pixel purity index algorithm on hyperspectral images," *Proc. Eng. Reconfigurable Syst. Algorithms Conf.*, 2010, pp. 125–131.

[60] C. Gonzalez, J. Resano, D. Mozos, A. Plaza, and D. Valencia, "FPGA implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis," *EURASIP J. Adv. Signal Process.*, 2010, article 969806.

[61] J. Boardman, "Automating spectral unmixing of AVIRIS data using convex geometry concepts," presented at the Summaries of Airborne Earth Science Workshop, 1993, JPL Publication 93-26, pp. 111–114.

[62] D. Valencia, A. Plaza, M. A. Vega-Rodríguez, and R. M. Pérez, "FPGA design and implementation of a fast pixel purity index algorithm for endmember extraction in hyperspectral imagery *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 5995, 2005, DOI: 10.1117/12.631270.

[63] J. Morales, N. Medero, N. G. Santiago, and J. Sosa, "Hardware implementation of image space reconstruction algorithm using FPGAs," in *Proc. 49th IEEE Int. Midwest Symp. Circuits Syst.*, 2006, vol. 1, pp. 433–436.

[64] C.-I. Chang, W. Xiong, and C.-C. Wu, "Field-programmable gate array design of implementing simplex growing algorithm for hyperspectral endmember extraction," *IEEE Trans. Geosci. Remote Sens.*, 2012, DOI: 10.1109/TGRS.2012.2207389.

[65] C.-I. Chang, C. Wu, W. Liu, and Y. C. Ouyang, "A growing method for simplex-based endmember extraction algorithms," *IEEE Trans. Geosci. Remote Sens.*, vol. 44, no. 10, pp. 2804–2819, Oct. 2006.

[66] C.-I. Chang, C. C. Wu, C.-S. Lo, and M.-L. Chang, "Real-time simplex growing algorithms for hyperspectral endmember extraction," *IEEE Trans. Geosci. Remote Sens.*, vol. 40, no. 4, pp. 1834–1850, Apr. 2010.

[67] W. Xiong, C.-C. Wu, C.-I. Chang, K. Kapalkis, and H. M. Chen, "Fast algorithms to implement N-FINDR for hyperspectral endmember extraction," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 545–564, Sep. 2011.

[68] S. Lopez, P. Horstrand, G. M. Callico, J. F. Lopez, and R. Sarmiento, "A novel architecture for hyperspectral endmember extraction by means of the Modified Vertex Component Analysis (MVCA) algorithm," *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.*, vol. 5, no. 6, pp. 1837–1848, Dec. 2012.

[69] S. Lopez, P. Horstrand, G. M. Callico, J. F. Lopez, and R. Sarmiento, "A Low-computational-complexity algorithm for hyperspectral endmember extraction: Modified vertex component analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 9, no. 3, pp. 502–506, May 2012.

[70] D. Hongtao and Q. Hairong, "An FPGA implementation of parallel ICA for dimensionality reduction in hyperspectral images," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2004, vol. 5, pp. 3257–3260.

[71] M. Lennon, G. Mercier, M. C. Mouchot, and L. Hubert-Moy, "Independent component analysis as a tool for the dimensionality reduction and the representation of hyperspectral images," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 4541, pp. 2893–2895, 2001.

[72] A. B. Lim, J. C. Rajapakse, and A. R. Omondi, "Comparative study of implementing ICNNs on FPGAs," in *Proc. Int. Joint Conf. Neural Netw.*, 2001, vol. 1, pp. 177–182.

[73] F. Sattar and C. Charayaphan, "Low-cost design and implementation of an ICA-based blind source separation algorithm," in *Proc. Annu. IEEE Int. ASIC/SOC Conf.*, 2002, pp. 15–19.

[74] A. Jacobs, C. Conger, and A. D. George, "Multiparadigm space processing for hyperspectral imaging," in *Proc. IEEE Aerosp. Conf.*, 2008, vol. 1, pp. 8–11.

[75] C. Chang, H. Ren, and S. Chiang, "Real-time processing algorithms for target detection and classification in hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 39, no. 4, pp. 760–768, Apr. 2001.

[76] C.-I. Chang, *Hyperspectral Imaging: Techniques for Spectral Detection and Classification.* New York: Kluwer Academic/Plenum, 2003.

[77] R. Nekovei and M. Ashtijou, "Reconfigurable acceleration for hyperspectral target detection," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2007, pp. 3229–3232.

[78] Q. Du and H. Ren, "Real-time linear constrained discriminant analysis to hyperspectral imagery," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 4548, pp. 103–108, 2001.

[79] S. Bernabe, S. Lopez, A. Plaza, R. Sarmiento, and P. G. Rodriguez, "FPGA design of an automatic target generation process for hyperspectral image analysis," in *Proc. IEEE Int. Conf. Parallel Distrib. Syst.*, 2011, pp. 1010–1015.

[80] K.-S. Park, S. H. Cho, S. Hong, and W.-D. Cho, "Real-time target detection architecture based on reduced complexity hyperspectral processing," *EURASIP J. Adv. Signal Process.*, pp. 1–18, Sep. 2008, article 438051.

[81] Q. Du and R. Nekovei, "Fast real-time onboard processing of hyperspectral imagery for detection and classification," *J. Real-Time Image Process.*, vol. 4, no. 3, pp. 273–286, Aug. 2009.

[82] Z. K. Baker, M. B. Gokhale, and J. L. Tripp, "Matched filter computation on FPGA, cell and GPU," in *Proc. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2007, pp. 207–218.

[83] Q. Du and J. E. Fowler, "Hyperspectral image compression using JPEG2000 and principal component analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 4, no. 2, pp. 201–205, Apr. 2007.

[84] N. R. Mat Noor, T. Vladimirova, and M. N. Sweeting, "High-performance lossless compression for hyperspectral satellite imagery," in *Proc. UK Electron. Forum*, 2010, pp. 78–83.

[85] C. Egho, T. Vladimirova, and M. N. Sweeting, "Acceleration of Karhunen-Loève transform for system-on-chip platforms," in *Proc. 7th NASA/ESA Conf. Adapt. Hardware Syst.*, 2012, pp. 272–279.

[86] N. R. Mat Noor and T. Vladimirova, "Integer KLT design space exploration for hyperspectral satellite image compression," *Lecture Notes in Computer Science*, vol. 6935. Berlin, Germany: Springer-Verlag, 2011, pp. 661–668.

[87] C. Egho and T. Vladimirova, "Hardware acceleration of Karhunen-Loeve transform for compression of hyperspectal satellite imagery," in *Proc. Australian Space Sci. Conf.*, 2011, pp. 237–248.

[88] P. Hao and Q. Shi, "Matrix factorizations for reversible integer mapping," *IEEE Trans. Signal Process.*, vol. 49, no. 10, pp. 2314–2324, Oct. 2010.

[89] M. Fleury, R. P. Self, and A. C. Downton, "Development of a fine-grained Karhunen-Loeve transform," *J. Parallel Distrib. Comput.*, vol. 64, no. 4, pp. 520–535, Apr. 2004.

[90] C. Egho and T. Vladimirova, "Hardware acceleration of the integer Karhunen-Loève transform algorithm for satellite image compression," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2012, pp. 4062–4065.

[91] M. E. Winter, "A proof of the N-FINDR algorithm for the automated detection of endmembers in a hyperspectral image," *Proc. SPIE—Int. Soc. Opt. Eng.*, vol. 5425, pp. 31–41, 2004.

[92] A. A. Green, M. Berman, P. Switzer, and M. D. Craig, "A transformation for ordering multispectral data in terms of image quality with implications for noise removal," *IEEE Trans. Geosci. Remote Sens.*, vol. 26, no. 1, pp. 65–74, Jan. 1988.

[93] R. A. Schowengerdt, *Remote Sensing: Models and Methods for Image Processing*. New York: Academic, 1997.

[94] J. Tabero, H. Mecha, J. Septién, S. Román, and D. Mozos, "A vertex-list approach to 2D Hw multitasking management in RTR FPGAs," in *Proc. DCIS Conf.*, 2003, pp. 545–550.

[95] S. Román, J. Septién, H. Mecha, and D. Mozos, "Constant complexity management of 2D HW multitasking in run-time reconfigurable FPGAs," *Lecture Notes in Computer Science*, vol. 3985. Berlin, Germany: Springer-Verlag, 2006, pp. 187–192.

[96] J. Resano, J. A. Clemente, C. González, D. Mozos, and F. Catthoor, "Efficiently scheduling runtime reconfigurations," *ACM Trans. Design Autom. Electron. Syst.*, vol. 13, no. 4, pp. 58–69, Sep. 2008.

[97] J. A. Clemente, C. González, J. Resano, and D. Mozos, "A task graph execution manager for reconfigurable multi-tasking systems," *Microprocess. Microsyst.*, vol. 34, no. 2–4, pp. 73–83, Mar. 2010.

[98] S. Lopez, G. M. Callico, A. Medina, J. F. Lopez, and R. Sarmiento, "High-level FPGA-based implementation of a hyperspectral Endmember Extraction Algorithm," in *Proc. Workshop Hyperspectral Image Signal Process., Evol. Remote Sens.*, 2012, pp. 161–165.

[99] T. Flatley, "SpaceCube: A family of reconfigurable hybrid on-board science data processors," presented at the NASA/ESA Conf. Adapt. Hardware Syst., Nuremberg, Germany, Jun. 25–28, 2012, Keynote Address I.

ABOUT THE AUTHORS

Sebastian Lopez (Member, IEEE) was born in Las Palmas de Gran Canaria, Spain, in 1978. He received the Electronic Engineer degree from the University of La Laguna, Santa Cruz de Tenerife, Spain, in 2001, obtaining regional and national awards for his CV during his degree, and the Ph.D. degree from the University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, in 2006.

Currently, he is an Associate Professor at the University of Las Palmas de Gran Canaria, developing his research activities at the Integrated Systems Design Division of the Institute for Applied Microelectronics (IUMA). He has published more than 50 papers in international journals and conferences. His research interests include real-time hyperspectral imaging systems, reconfigurable architectures, video coding standards, and hyperspectral image compression systems.

Prof. Lopez is a member of the IEEE Geoscience and Remote Sensing Society and the IEEE Consumer Electronics Society as well as a member of the Publications Review Committee of the IEEE Transactions on Consumer Electronics. Additionally, he currently serves as an active reviewer of the IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, the IEEE Geoscience and Remote Sensing Letters, the IEEE Transactions on Circuits and Systems for Video Technology, the *Journal of Real Time Image Processing*, *Microprocessors and Microsystems: Embedded Hardware Design* (MICPRO), and the IET *Electronics Letters*.

Tanya Vladimirova (Member, IEEE) received the M.Sc. degree in applied mathematics from the Technical University of Sofia, Sofia, Bulgaria, in 1980 and the M.Eng. degree in computer systems engineering and the Ph.D. degree in very large scale integration (VLSI) design from the St. Petersburg Electro-Technical University (LETI), St. Petersburg, Russia, in 1978 and 1984, respectively.

Currently, she is a Professor of Embedded Systems and VLSI Design at the Department of Engineering, University of Leicester, Leicester, U.K. She was with the Department of Electronic Engineering, University of Surrey, Surrey, U.K., and led the VLSI Design and Embedded Systems research group at the Surrey Space Centre. Her research interests are in the areas of low-power onboard integrated circuit design, image processing, intelligent embedded systems and space-based wireless sensor networks.

Prof. Vladimirova acted as a Co-Chair of the NASA conference on Military and Aerospace Applications of Programmable Logic Devices (MAPLD) from 2000 to 2006.

Carlos González received the M.S. and Ph.D. degrees in computer engineering from the Complutense University of Madrid, Madrid, Spain, in 2008 and 2011, respectively.

Currently, he is a Teaching Assistant in the Department of Computer Architecture and Automation, Universidad Complutense Madrid. As a research member of GHADIR group, he mainly focuses on applying runtime reconfiguration in aerospace applications. His research interests include remotely sensed hyperspectral imaging, signal and image processing, and efficient implementation of large-scale scientific problems on reconfigurable hardware. He is also interested in the acceleration of artificial intelligence algorithms applied to games.

Dr. González won the Design Competition of the 2009 and 2010 IEEE International Conferences on Field Programmable Technology (FPT). He received the Best Paper Award of an Engineer under 35 years old at the 2011 International Conference on Space Technology.

Javier Resano received the B.S. degree in physics, the M.S. degree in computer science, and the Ph.D. degree in computer science from the Universidad Complutense of Madrid, Madrid, Spain, in 1997, 1999, and 2005, respectively.

Currently, he is an Associate Professor at the Computer Engineering Department, University of Zaragoza, Zaragoza, Spain, and he is a member of the GHADIR research group, from Complutense University of Madrid, Madrid, Spain, and the GAZ research group, from the University of Zaragoza. He is also a member of the Engineering Research Institute of Aragon (I3A). His research has been focused on hardware/software codesign, task scheduling techniques, dynamically reconfigurable hardware, and FPGA design.

Prof. Resano received several international awards including the first prize at the Design Competition of the 2009 and 2010 IEEE International Conferences on Field Programmable Technology (FPT) for his FPGA designs.

Daniel Mozos received the B.S. degree in physics and the Ph.D. degree in computer science from the Complutense University of Madrid, Madrid, Spain, in 1986 and 1992, respectively.

He is a permanent Professor in the Department of Computer Architecture and Automatics, Complutense University of Madrid, where he leads the GHADIR research group on dynamically reconfigurable architectures. His research interests include design automation, computer architecture, and reconfigurable computing.

Antonio Plaza (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer engineering from the University of Extremadura, Extremadura, Spain, in 1999 and 2002, respectively.

He has been a Visiting Researcher with the Remote Sensing Signal and Image Processing Laboratory (RSSIPL), University of Maryland Baltimore County, Baltimore; with the Applied Information Sciences Branch, NASA Goddard Space Flight Center (GSFC), Greenbelt, MD; with the Airborne Visible Infrared Imaging Spectrometer Data Facility, NASA Jet Propulsion Laboratory (JPL), Pasadena, CA; with the Telecommunications and Remote Sensing Laboratory, University of Pavia, Pavia, Italy; and with the GIPSA-lab, Grenoble Images Parole Signal Automatique, Grenoble, France. He is currently an Associate Professor (with accreditation for Full Professor) with the Department of Technology of Computers and Communications, University of Extremadura, where he is the Head of the Hyperspectral Computing Laboratory (HyperComp). He was the Coordinator of the Hyperspectral Imaging Network, a European project designed to build an interdisciplinary research community focused on hyperspectral imaging activities. He is the author or coauthor of more than 350 publications on remotely sensed hyperspectral imaging, including more than 90 journal citation report papers (45 since January 2011), around 20 book chapters, and over 230 conference proceeding papers. His research interests include remotely sensed hyperspectral imaging, pattern recognition, signal and image processing, and efficient implementation of large-scale scientific problems on parallel and distributed computer architectures.

Dr. Plaza has guest edited seven special issues in scientific journals on the topic of remotely sensed hyperspectral imaging. He has been a Chair for the IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (Whispers 2011). He has also been serving as a Chair for the SPIE Conference on Satellite Data Compression, Communications, and Processing, since 2009, and for the SPIE Europe Conference on High-Performance Computing in Remote Sensing, since 2011. He has been a recipient of the recognition of Best Reviewers of the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS in 2009 and a recipient of the recognition of Best Reviewers of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING in 2010, a journal for which he has served as an Associate Editor since 2007 and for which he has reviewed more than 260 manuscripts. He is currently serving as the Director of Education Activities for the IEEE Geoscience and Remote Sensing Society (GRSS), and as President of the Spanish Chapter of IEEE GRSS. He has been appointed Editor of the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, with a three-year term starting in January 2013.