Hyperspectral Compressive Sensing With a System-On-Chip FPGA

José M. P. Nascimento, Mário P. Véstias, and Gabriel Martín

***Abstract*—Advances in hyperspectral sensors have led to a significantly increased capability for high-quality data. This trend calls for the development of new techniques to enhance the way that such unprecedented volumes of data are stored, processed, and transmitted to the ground station. An important approach to deal with massive volumes of information is an emerging technique, called compressive sensing, which acquires directly the compressed signal instead of acquiring the full dataset. Thus, reducing the amount of data that needs to be measured, transmitted, and stored in first place. In this article, a hardware/software implementation in a system-on-chip (SoC) field-programmable gate array (FPGA) for compressive sensing is proposed. The proposed hardware/software architecture runs the compressive sensing algorithm with a unitary compression rate over an airborne visible/infrared imaging spectrometer sensor image with 512 lines, 614 samples, and 224 bands in 0.35 s. The proposed system runs 49× and 216× faster than an embedded 256-cores GPU of a Jetson TX2 board and the ARM of the SoC FPGA, respectively. In terms of energy, the proposed architecture requires around 100× less energy.**

***Index Terms*—Compressive sensing, field-programmable gate arrays (FPGA), hyperspectral imagery, on-board processing, real time.**

I. Introduction

HYPERSPECTRAL sensors acquire images containing hundreds of spectral data bands with high spatial and spectral resolution. The high spectral resolution of these sensors allows an accurate identification of the different materials contained in the scene of interest. This feature among others, has turn hyperspectral images into a powerful tool in many applications in the fields of agriculture [1], surveillance [2], [3], medical imaging [4], food safety [5], [6], forensic applications [7], [8], and many others [9].

Manuscript received December 23, 2019; revised March 17, 2020 and May 11, 2020; accepted May 14, 2020. Date of publication May 28, 2020; date of current version July 6, 2020. This work was supported in part by Instituto de Telecomunicações and Fundação para a Ciência e a Tecnologia (FCT) under Project UID/EEA/50008/2019 and Project FIREFRONT with reference PCIF/SSI/0096/2017, and in part by National funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UIDB/50021/2020. *(Corresponding author: José M. P. Nascimento.)*

José M. P. Nascimento is with the Instituto Superior de Engenharia de Lisboa, 1959-007 Lisbon, Portugal, and also with the Instituto de Telecomunicações, 1049-001 Lisbon, Portugal (e-mail: zen@isel.pt).

Mário P. Véstias is with the Instituto Superior de Engenharia de Lisboa, 1959-007 Lisbon, Portugal, and also with the Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento, 1000-029 Lisbon, Portugal (e-mail: mvestias@deetc.isel.ipl.pt).

Gabriel Martín is with the Instituto de Telecomunicações, 1049-001 Lisbon, Portugal (e-mail: g.martin.he@gmail.com).

Digital Object Identifier 10.1109/JSTARS.2020.2996679

Considering the collected data in a 2-D spatial domain of megapixel size and with the spectral dimension with hundreds of bands, one can represent the data as a 3-D image cube comprising a huge amount of data. Consequently, its scanning, storage, and digital processing is challenging [10]. In remote sensing scenarios where hyperspectral images are collected on-board satellites and need to be transferred to the Earth's ground station, an efficient compression of such images is mandatory [11].

The compressed sensing (CS) theory proposed in [12] and [13] has received considerable interest since it states that if a signal is sparse itself, thus it can be sampled with much less data than those dictated by the Shannon–Nyquist theorem and reconstructed accurately with these sampled data [14], [15]. Fortunately, the structure of hyperspectral data is sparse [16], [17] and it can be modeled by a linear mixing model, considering that the total set of pixel vectors are represented by a few number of endmembers [9], [18]. Additionally, these images also present a high correlation in the spatial domain, which may improve the compression ratio and the quality of the reconstructed image [19]. These hyperspectral features have encouraged recent developments and implementations of CS techniques on hyperspectral imagery [20]–[26].

Running compressive sensing algorithms in on-board processing platforms is subject to throughput and power constraints. Images are acquired at a certain rate, and therefore, these platforms must be fast enough for real-time processing to avoid image storage. For example, the AVIRIS senses 512 pixels of 224 spectral bands in 8.3 ms. So, 614 samples must be processed in about 5 s. On-board processing is also subject to power constraints. Thus, platforms must be designed for best energy efficiency with reduced power.

Since CS measurement process is based on performing a large number of parallel dot products between random vectors and the signal of interest, graphics processing units (GPUs) are well suited to perform this task. Several implementations of CS and random projections algorithms over hyperspectral data with GPUs have been proposed [27]–[29], concluding that by using GPUs, it is possible to achieve real-time performance for the random projection step. On the other hand, the power requirements of this hardware make them ineffective for on-board applications. Over the last years, the advances in the semiconductor industry and the huge interest in developing mobile devices have allowed companies such as NVIDIA to develop low power GPUs. For example, Jetson TX2 board has a low power consumption GPU, that nevertheless, can achieve

high throughput in image processing applications at the same time [11], [30], [31].

Field-programmable gate arrays (FPGAs) are also a good platform for on-board processing systems since they have high computational performance, compact size, reduced weight, and low power consumption among other characteristics. Additionally, FPGAs permit the adaptation of the hardware to the needs of different missions, which make them appealing for satellite platforms. [32]–[36]. However, in order to include FPGA in satellite payload, they must be resistant to damages or malfunctions caused by ionizing radiation, present in the harsh environment of outer space [37]–[39]. The available rad-hard FPGAs (e.g., Virtex-5QV) easily provide sufficient resources to implement the proposed architecture with the same performance and with fault tolerance.

In this article, a hardware/software architecture is proposed to run the CS method. The architecture is implemented in a system-on-chip (SoC) FPGA. The performance of the system is compared to the execution of the algorithm in the ARM of the SoC FPGA and in an embedded GPU on a Jetson TX2 board. The work compares the performance and the power consumption of the three implementations. The results indicate that the proposed system delivers a peak performance of 96.8 GOPs (Giga Operations per second). and runs the compressive sensing algorithm with a unitary compression rate over an AVIRIS sensor image with 512 lines, 614 samples, and 224 bands in 0.35 s. Compared to the other two platforms, it runs 49 times and 216 times faster than the embedded GPU and the ARM, respectively. In terms of energy, the proposed architecture requires around 100 times less energy than the other two solutions.

The rest of this article is organized as follows. In Section II, the compressive sensing method is summarized. Section III describes the design and implementation of the proposed hardware/software architecture. In Section IV, a set of experiments are conducted to demonstrate the effectiveness of the architecture to execute compressed sensing. Also, the architecture is compared to other computing platforms. Finally, Section V concludes this article and presents future lines of research work.

II. Compressive Sensing Method

In this section, the CS method termed hyperspectral coded aperture (HYCA) [22] is briefly described. This method for its characteristics is well suited to be developed in a parallel fashion [27]. It takes advantage of the following two central properties of most hyperspectral images:

1) the spectral vectors live systematically in low-dimensional subspaces [17];
2) the spectral bands present a high correlation in the spatial domain.

The former property allows to represent the data vectors using a reduced set of spectral endmembers due to the mixing phenomenon [18] and also exploits the high spatial correlation of the fractional abundances associated to the spectral endmembers. HYCA performs CS in the spectral domain, for this purpose, a set of q inner products between random vectors and the image pixels is performed, with q lower than the original number of bands of the hyperspectral data $bands$. Thus, the size of the compressed signal is $\frac{\text{bands}}{q}$ times smaller than the original. This operation may be represented as $\mathbf{y}_p = \mathbf{H}_p\mathbf{x}_p$ for $p \in \{1, \ldots, n\}$, where n is the number of pixels of the image, $\mathbf{y}_p \in \mathbb{R}^q$ is the pth compressed pixel, $\mathbf{H}_p$ is a matrix containing the random vectors used for the measurement process for the pth pixel, which is represented as $\mathbf{x}_p \in \mathbb{R}^{bands}$. Due to the fact that the number of pixels n in a given scene may be very large, for instance, an AVIRIS sensor acquire for each image scene a set of 512 scans containing 614 samples and 224 bands, which yields approximately 140 MB. Thus, storing in memory different matrices $\mathbf{H}_p$ for $p \in \{1, \ldots, n\}$ is unattainable, the HYCA measurement strategy splits the dataset into different windows of size $m = ws \times ws$, and then, repeat the matrices $\mathbf{H}_i$ used in each window, thus requiring to store in memory just m different $\mathbf{H}_p$ matrices. Formally, the pseudocode of the compressive sensing method is given by Algorithm 1.

Algorithm 1: Compressive Sensing Algorithm.

Input: X - 3D Hyperspectral image with $(lines \times samples \times bands)$ pixels;
H - Matrix of random vectors;
ws - Window size
Result: Y - Compressed Image of size $lines \times samples \times q$

```
for j ← 0 to lines − 1 do
    il ← j mod ws;
    for i ← 0 to samples − 1 do
        ic ← i mod ws;
        for n ← 0 to q − 1 do
            Y[n + (j × samples + i) × q] ←
              Σ_{b=0}^{bands−1} H[b + n × bands + (il × ws +
              ic) × bands × q] × X[b + (j × samples +
              i) × bands]
        end
    end
end
```

Algorithm 1 considers that the matrix $\mathbf{X}$ is represented as a 2-D matrix of size lines × samples, where each entry of the matrix (pixel) is a vector of bands and that the disposition on memory follow the Band-Interleaved-by-Pixel format. For each pixel $\mathbf{X}(i, j)$, q inner products are calculated and accumulated between its vector of bands and a vector of matrix $\mathbf{H}$. The result is stored in the matrix $\mathbf{Y}$. Each inner product with a vector of $\mathbf{H}$ produces one element of the vector associated with the compressed pixel.

Considering that the measurements are sent from the on-board platform, the bulk of the processing to reconstruct the original image is performed on the Earth's ground station. The reconstruction of the original image can be formulated as an optimization problem, where it is assumed that the dataset live in a low-dimensional subspace [17]. Furthermore, the abundances exhibit a high spatial correlation and must be nonnegative, these

features are exploited for estimating $\mathbf{z}$ using the following:

$$\min_{\mathbf{z}\geq\mathbf{0}} (1/2)\|\mathbf{y}-\mathbf{H}\mathbf{z}\|^2 + \lambda_{\text{TV}}\text{TV}(\mathbf{z}). \quad (1)$$

Therefore, the minimization of (1) aims at finding a solution that is a compromise between the fidelity to the measured data, enforced by the quadratic term $(1/2)\|\mathbf{y}-\mathbf{K}\mathbf{z}\|^2$, and the properties enforced by the total variation regularizer $\text{TV}(\mathbf{z})$, that is piecewise smooth image of abundances. The relative weight between the two characteristics of the solution is set the regularization parameter $\lambda_{\text{TV}} > 0$.

To solve the convex optimization problem in (1), a set of new variables per term of the objective function were used and the ADMM methodology [40] has been adopted to decompose very hard problems into a cyclic sequence of simpler problems. Further details on the algorithm implementation and its parallelization can be found in works [27], [41].

III. Design and Implementation of the Hardware/Software Architecture

In this section, the hardware/software architecture to run the compressive sensing algorithm is described. The methodology followed to design the hardware architecture consisted of the following steps.

1) *Algorithm optimization:* The algorithm was reorganized to improve data accesses from external memory. Loop tilling exploits spatial and temporal locality allowing data to be accessed in blocks (tiles) permitting to execute the operations over a block of data stored in on-chip memory.
2) *Architecture design:* Design of a hardware/software architecture where the hardware runs the compute intensive operations of the algorithm and the processor controls the cycles of the algorithm and data transfers from/to external memory.

A. Algorithm Optimization

The original algorithm slides sequentially over the input pixels of the hyperspectral image. The problem with this approach is that the matrix H is not reused by the next pixel. So, unless the local memory is enough to hold all ws × ws $\mathbf{H}$ matrices, each $\mathbf{H}$ matrix is read $\frac{\text{lines}\times\text{samples}}{\text{ws}\times\text{ws}}$ times from main memory. This introduces a penalty in the execution time of the algorithm.

Since there are no data dependencies between the calculation of different output pixels and there are no constraints over the order with each output pixels must be produced, a loop tilling technique has been applied to the algorithm that guarantees that each matrix $\mathbf{H}$ is only read once from the main memory (see Algorithm 2).

In the optimized algorithm, each different matrix $\mathbf{H}$ is only read once and used in all pixels multiples of the window size. Each matrix $\mathbf{H}$ is then reused $\frac{\text{lines}}{\text{ws}} \times \frac{\text{samples}}{\text{ws}}$ times before the next matrix $\mathbf{H}$ is read. Also, each pixel is reused q times in the calculation of the inner products between the pixel and the vectors of matrix $\mathbf{H}$.

Algorithm 2: Compressive Sensing Algorithm with Optimization of Memory Accesses.

Input: X - 3D Hyperspectral image with $(lines \times samples \times bands)$ pixels
H - Matrix of random vectors;
ws - Window size
Result: Y - Compressed Image of size $lines \times samples \times q$

```
1  for r ← 0 to ws − 1 do
2      for k ← 0 to ws − 1 do
3          for m ← 0 to q × bands − 1 do
4              H_local[m] ← H[m+(r×ws+k)×bands×q]
5          end
6          for j ← r to lines-1, j = j + ws do
7              for i ← k to samples-1, i = i + ws do
8                  for l ← 0 to bands-1 do
9                      X_local[l] ←
                         X[l + (j × samples + i) × bands]
10                 end
11                 for n ← 0 to q-1 do
12                     Y[n + (j × samples + i) × q] ←
                         Σ_{b=0}^{bands−1} H_local[b + n × bands] ×
                         X_local[b]
13                 end
14             end
15         end
16     end
17 end
```

B. FPGA Architecture

This section presents the proposed hardware/software implementation of Algorithm 2 described in the previous section. The architecture is designed to support real-time processing of hyperspectral images acquired from the AVIRIS sensor. AVIRIS is a whiskbroom scanning system that collects data in a 12-bit quantization. Each image contains 614 × 512 pixels comprising 224 spectral bands in the range from 370 to 2500 nm.[1] Radiance values are stored, after onboard calibration, with 16-bit integers [42]. Thus, the proposed architecture use 16 bits short integer (int16) that guarantees enough precision for the algorithm.

The architecture consists of a general-purpose processor (*ARM*) and a dedicated hardware accelerator to run the core of the algorithm. Algorithm 2 was partitioned into the accelerator and the processor. The processor controls the whole algorithm, namely the cycles in lines 1, 2, 6, and 7 of the algorithm. In lines 3–5 and lines 8–10, the processor configures a set of direct memory access (DMAs) to send matrix $\mathbf{H}$ and the input image to the hardware. Lines 11–13 are implemented in a hardware consisting of the inner product calculation. The compressed image is sent back to the main memory. The block diagram of the architecture is illustrated in Fig. 1.

The architecture contains an ARM processor with access to external memory and to the accelerator implemented in the

[1][Online] Available: https://aviris.jpl.nasa.gov/html/aviris.instrument.html

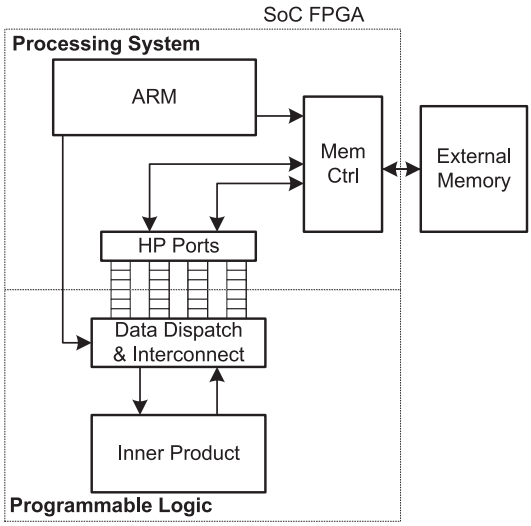


Fig. 1. Hardware/Software architecture designed to run the compressive sensing algorithm.

programmable logic of the FPGA. The transfer of data between the external memory and accelerator in the programmable logic is done through four high-performance (HP) ports using four DMA blocks. The *data dispatch and interconnect* (DPI) block is responsible for forwarding data between DMA buffers and on-chip memories used to store the matrices of the algorithm to be processed by the inner product module. Block DPI is configured and controlled by the processor.

The architecture can be implemented in a non-SoC FPGA replacing the processor of the SoC FPGA by a soft processor or a dedicated controller for a fixed algorithm configuration.

The *inner product* block implements the inner products of the algorithm (lines 11–13). To improve the performance of the solution, multiple values are read from local memories and calculated in parallel (see Fig. 2).

All bands of a pixel can be read and multiplied by an $\mathbf{H}$ vector in parallel. More parallelism can be exposed by unrolling the cycle of line 11 (parallel inner products for different q). The inner product block is statically configurable in terms of parallel multipliers and unrolling factor, permitting to optimize the architecture for a required throughput and available FPGA resources. When unrolled, multiple inner products are calculated in parallel, one for each value of q. The example illustrated in Fig. 2 and later used in the tested implementations, corresponds to an implementation of the cycle unrolled four times.

The output of the multipliers are accumulated with an adder tree. The adder tree guarantees full arithmetic, that is, adders of each subsequent level have an extra bit to represent the result to keep full precision. The number of levels of the adder tree, l, depends on the number of multipliers, $Nmult$, that is: $l = \lceil \log_2 Nmult \rceil$. The accumulator (ACC) in the end of the adder tree is required if the number of multipliers is lower than the number of bands. In this case, the inner product must be executed in multiple steps and the intermediate results are accumulated in ACC. For example, to process 224 bands with only 112 multipliers, the architecture determines the inner product of the first 112 bands and the result is accumulated with the inner product of the next 112 bands. Therefore, in this case, it takes two steps. To improve the throughput of the circuit, the whole datapath is pipelined, illustrated with gray lines (registers) in the figure.

After calculating the inner products, the results are truncated back to 16 bits before being stored and sent to the external memory.

The **DDI** block transfers data (H matrix, image) from the external memory to on-chip memories (H memory and pixel memory) and from the *Y* memory (compressed image) to external memory (see Fig. 3).

The data transfer is done through the four HP ports of the ZYNQ FPGA that allow a total data transfer of up to (4×1.2 GB/s). Four DMAs are used to do data transfers. These are dynamically configured for specific data transfers (start address and data size) by a central controller that is configured by the ARM processor. Each on-chip memory has an associated address generator that generates read and write addresses of simple dual port memories. On-chip memories are dual port to allow simultaneous read and write of data.

All four DMAs are used to read data from the external memory to the on-chip memory. One DMA is used to transfer the compressed image from the local memory (Y memory) to the external memory.

The H and pixel memories store four bands of four different H vectors and pixels, respectively, in each memory write and the Y memory stores four bands of an output pixel in a single write. H and pixel memories must have a large bandwidth so that multiple values are read in parallel. Therefore, these memories are implemented with a set of distributed memories (BRAMs of the FPGA) each having an output datawidth of 64 bits (see Fig. 4).

Each memory block H stores several vectors of the $\mathbf{H}$ matrix and each memory block $PixelMem$ stores several pixels.

The main control block of the DDI block guarantees the synchronization between data communication and computation. Following Algorithm 2 described previously, the ARM sets the control block to configure the DMAs to transfer the $\mathbf{H}$ matrix and the set of pixels to the local memories of the architecture. After transferring the first data, the controller signals the address generators and the inner product block to start execution. At the same time, configures the DMAs to transfer the next $\mathbf{H}$ matrix and the next set of pixels of the input image.

After finishing the operation, the inner product block notifies the controller. If the next $\mathbf{H}$ matrix and the first pixels for the next inner product are already available in the on-chip memories, the controller signals the address generators to restart again. The process repeats until the end of the algorithm.

IV. Experimental Results

The proposed hardware architecture has been described in VHDL and implemented on a Xilinx Zynq Zedboard with a XC7Z020 SoC FPGA. The hardware design and implementation has been done with Vivado Design Suite 2019.1 and the power

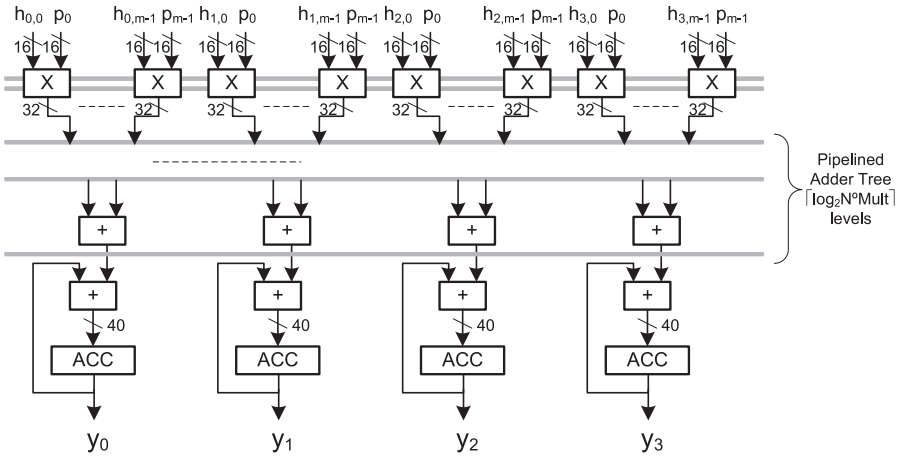


Fig. 2. Architecture of the inner product block of the proposed architecture.

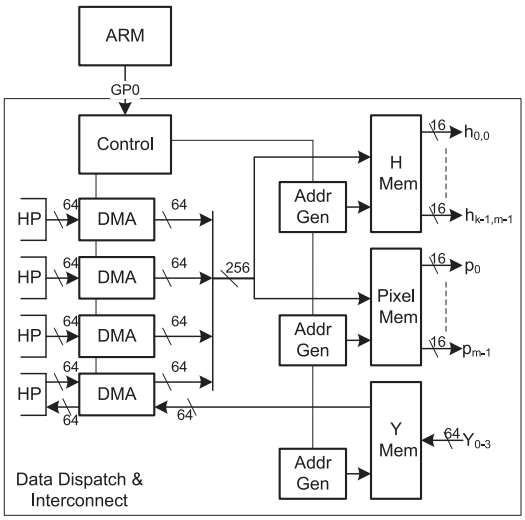


Fig. 3. Block diagram of the DDI block.

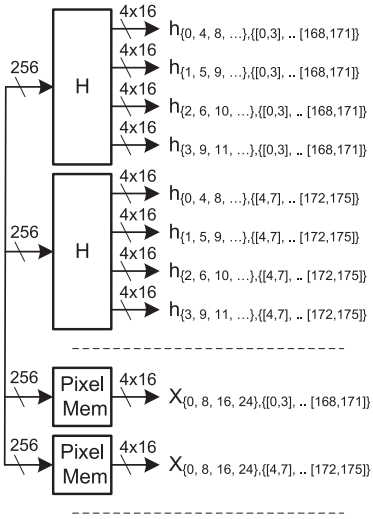


Fig. 4. Organization of the on-chip memories: H and Pixel memories.

of the circuits has been estimated with Xilinx Power Estimator tool.

The FPGA board has 512-MB DDR3 memory with a measured 3.3 GB/s of memory bandwidth. This is the memory bandwidth available to transfer the hyperspectral image and matrices $\mathbf{H}$ to the FPGA and the compressed image from the FPGA to the external memory.

The target FPGA is an SoC that contains a Dual ARM Cortex-A9 and a reconfigurable area with Artix-7 technology. This family of FPGAs is quite appropriate to develop embedded systems making these boards ideal for fast prototyping, proof-of-concept development, and fast deployment of embedded systems. The programmable logic of the FPGA has 85 K logic cells with 106 400 registers, 53 200 lookup tables (LUTs), 140 BRAMs, and 220 digital signal processing blocks (DSP48).

To test the architecture, the DDR memory available in the board was utilized to store the dataset. The ARM processor available in the FPGA was utilized as the processor of the proposed architecture.

The experiments are carried out on the Cuprite AVIRIS scene labeled as f970619t01p02_r02_sc03.a.rfl. This scene has 614×512 pixels comprising 224 spectral bands.

A. Area, Performance, and Power of the Proposed Architecture

Several implementations of the proposed architecture were designed for different compression rates and with different number of multipliers in a pipelined datapath to calculate four output pixels in parallel (unroll factor of four). The utilization of resources after postplace and route is given in Table I.

The number of multipliers determines the number of used DSPs, while q determines the number of BRAMs of the architecture. The largest architecture, with the higher critical path, operates at a maximum frequency of 220 MHz. The same operating frequency was considered for all architectures.

The architecture with 224 multipliers was implemented for different compression rates and the execution times to compress

TABLE I
UTILIZATION OF RESOURCES OF THE PROPOSED ARCHITECTURE FOR DIFFERENT CONFIGURATIONS IN TERMS OF NUMBER OF MULTIPLIERS OF THE INNER PRODUCT BLOCK AND FOR SEVERAL VALUES OF q

# Multipliers	LUTs	DSPs	BRAMs q:224, 112, 48, 16
56×4	15188	220	135, 135, 135, 135
28×4	10253	112	107, 79, 79, 79
14×4	8430	56	93, 65, 51, 51
7×4	7452	28	86, 58, 44, 36
4×4	7001	16	83, 55, 41, 30

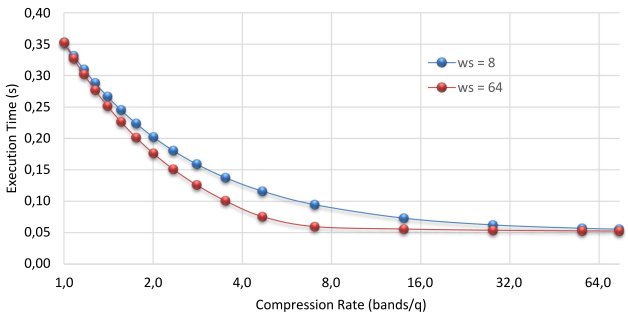


Fig. 5. Execution time of the compression process of a hyperspectral image ($512 \times 614 \times 224$) for different compression rates and window sizes of 8 and 64. The circuit runs at 220 MHz in a ZYNQ7020 SoC FPGA.

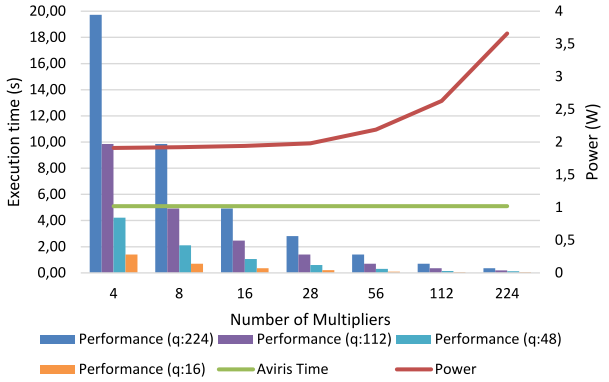


Fig. 6. Execution time of the compression process of a hyperspectral image ($512 \times 614 \times 224$), for different values of q, and for different levels of architectural parallelism. The circuit runs at 220 MHz in a ZYNQ7020 SoC FPGA.

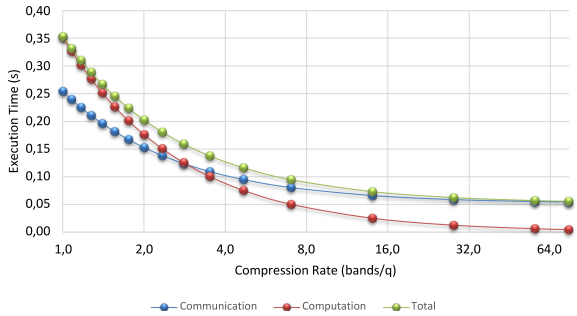


Fig. 7. Computation versus communication time of the compression process of a hyperspectral image ($512 \times 614 \times 224$), for different compression rates and ws = 64. The circuit runs at 220 MHz in a ZYNQ7020 SoC FPGA.

an hyperspectral image of size $512 \times 614 \times 224$ were determined (see the results for window sizes of 8 and 64 in Fig. 5).

With a unitary compression, the proposed circuit compresses an hyperspectral image of size $512 \times 614 \times 224$ in 0.35 s. The execution time reduces five times with a compression rate of 14. The circuit has a total power of 3.66 W. Another important observation is that the execution time of the compression algorithm in our architecture reduces with the size of the window. The figure shows the results for the extreme cases of window sizes: 8 and 64. The largest variations occur for the higher compression rates.

The previous performance results are for a configuration with the highest throughput. However, since the AVIRIS sensor acquires 512 pixels of 224 spectral bands in 8.3 ms [43] (5.1 s to process 614 samples), the parallelism can be reduced, which reduces the required hardware resources and the power (see Fig. 6).

As can be observed, with just 16 multipliers in parallel, with $q = 224$ and a power 1.94 W, it is possible to execute AVIRIS images in real time. This design uses just 7001 LUTs, 16 DSPs, and 31 BRAMs. The power is smaller but the energy increases from 1.3 to 9.6 J, since the power associated with the processor is almost constant for different hardware architectures. With compression rates above four, the results show that four multipliers are enough to run the algorithm in real time. Consequently, the hardware resources are drastically reduced.

The compression rate determines the requirements in terms of computation and communication. Therefore, the architecture can be optimized in terms of resources considering the compression rate. The execution times of both components, communication and computation, have been determined (see Fig. 7).

For low compression rates, the total execution time is determined by the computational performance of the architecture. For compression rates higher than three, the total execution time is determined by the communication performance of the architecture. In this case, the bottleneck is associated with the memory bandwidth.

Considering the communication to computation ratios, in the design of architecture, it is important to determine how efficient the computational resources are used. The metric to quantify this is performance efficiency, which is the ratio between the measured performance and peak performance converted to percentage.

To analyze the efficiency of the proposed architecture, three different architectures with different tradeoffs between the performance and performance efficiency were implemented for a window size of 64. The architectures have 224, 112, and 56 parallel multipliers. In all cases, we measured the execution times for different compression ratios, and from these, the performance efficiency was determined (see Fig. 8).

As can be observed from the figure, the performance efficiency reduces considerably (from 91% to 31%) with the compression rate when the architecture is designed with 224 multipliers. This is because the reduction in the compression rate increases the ratio between communication delay and computation delay. When the communication delay is higher than the computation delay, the idle times of computational units increase, and consequently, the performance efficiency reduces.

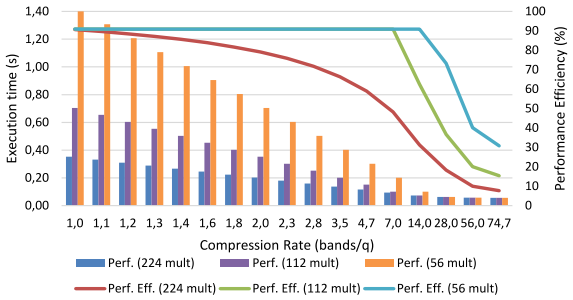


Fig. 8. Performance and performance efficiency of the compression process of a hyperspectral image ($512 \times 614 \times 224$), for different compression ratios and three different architectures (with 224, 112, and 56 multipliers) running at 220 MHz in a ZYNQ7020 SoC FPGA.

When the number of multipliers is reduced to half, the computation time doubles reducing the communication to computation ratio. With 112 multipliers, the performance efficiency reduces to 61% with a compression ratio of 14 up to 20% with the highest compression rates. The increase in the efficiency is traded off by performance, that is, improving the efficiency reduces the performance since there are less computational resources. With 56 multipliers, the highest performance efficiency is kept until a compression rate of 14. Since the execution times of the circuit with a compression rate above 14 are independent of the number of multipliers, the reduction in performance efficiency is due to the communication bottleneck.

The most appropriate architecture depends on the application requirements in terms of the required performance and energy. The designer should always try to be as close to the requirement as possible to improve the efficiency of the architecture.

B. Comparison With Other Embedded Computing Platforms

The performance, power, and energy consumption of the proposed SoC architecture was compared to other embedded computing platforms, namely the embedded GPU of the Jetson TX2 platform [44] and the ARM processor of the ZYNQ7020 SoC FPGA, in the execution of the compressive sensing algorithm.

Jetson TX2 incorporates a quad-core 2.0-GHz 64-bit ARMv8 A57 processor, a dual-core 2.0-GHz superscalar ARMv8 Denver processor, and an integrated embedded low-power Pascal GPU. There are two 2-MB L2 caches, one shared by the four A57 cores and one shared by the two Denver cores. The GPU has two streaming multiprocessors, each providing 128 1.3-GHz cores that share a 512-kB L2 cache. The six CPU cores and integrated GPU share 8 GB of 1.866-GHz DRAM memory [45]. The Jetson TX2 typically draws between 7.5 and 15 watts with a voltage input of 5.5–19.6 V dc and requires minimal cooling and additional space.

The processing system side of the ZYNQ7020 device contains a dual-core ARM Cortex-A9 working with a frequency of 667 MHz. The memory hierarchy consists of 32 kB level-1 cache for each core and 512 kB level-2 cache common to both cores, 256 kB of the on-chip memory and a memory controller to access the external board memory with a measured memory bandwidth of 3.3 GB/s. The dual-core ARM and caches are integrated in a complete processing system that also includes a NEON media processing engine and a single and double precision vector floating-point unit. The NEON engine was not used to run the algorithm.

TABLE II
COMPARISON OF THE DELAY OF PROPOSED ARCHITECTURE AGAINST AN EMBEDDED GPU AND A DUAL-CORE ARM PROCESSOR RUNNING COMPRESSIVE SENSING OVER AN AVIRIS SENSOR IMAGE WITH 512 LINES, 614 SAMPLES, AND 224 BANDS AND A WINDOW SIZE OF 64

Compression Rate	Embedded GPU (s)	ARM (s)	This work (s)
1.0	17.1	75.8	0.35
1.1	15.2	71.2	0.33
1.2	14.0	65.3	0.31
1.3	12.8	60.3	0.29
1.4	11.9	55.4	0.27
1.6	10.8	50.2	0.25
1.8	9.1	45.3	0.22
2.0	8.3	40.1	0.20
2.3	7.1	35.0	0.18
2.8	5.9	29.8	0.16
3.5	5.0	24.7	0.14
4.7	4.1	19.6	0.12
7.0	1.9	14.6	0.09
14.0	1.1	9.9	0.07

TABLE III
COMPARISON OF THE ENERGY OF THE PROPOSED ARCHITECTURE AGAINST AN EMBEDDED GPU AND A DUAL-CORE ARM PROCESSOR RUNNING COMPRESSIVE SENSING OVER AN AVIRIS SENSOR IMAGE WITH 512 LINES, 614 SAMPLES, AND 224 BANDS AND A WINDOW SIZE OF 64

Compression Rate	Embedded GPU (J)	ARM (J)	This work (J)
1.0	154.7	128.9	1.3
1.1	135.0	120.9	1.2
1.2	126.0	111.0	1.1
1.3	114.4	96.6	1.0
1.4	103.2	88.5	0.9
1.6	91.3	80.3	0.9
1.8	73.8	72.3	0.7
2.0	65.6	64.2	0.7
2.3	56.7	52.5	0.6
2.8	48.6	44.9	0.5
3.5	40.0	37.1	0.4
4.7	32.0	29.4	0.3
7.0	15.6	22.1	0.2
14.0	7.7	14.6	0.1

All platforms run the integer version of the algorithm for a fair comparison. Running the algorithm with integer data in these platforms achieves a slightly better performance than running with the floating-point arithmetic. This may be caused by improved utilization of cache or better compiler optimization.

The real hyperspectral dataset, acquired by the AVIRIS sensor, used in this experiments has 614 samples times 512 lines and 224 bands. The window size is set to 64 (see results in Table II).

The results show that the proposed hardware/software architecture is 49 times faster than the implementation in the

TABLE IV
COMPARISON OF THE PROPOSED ARCHITECTURE AGAINST CCSDS-123 IMPLEMENTATIONS

Work	FPGA Device	Freq (MHz)	Throughput (Mb/s)	Power (W)	Compression	LUTs	DSPs	BRAMs
Santos et al. [46]	Virtex5 FX130T	140	2240	—	3.45	4645	11	74
Tsigkanos et al. [47]	Virtex5 FX130T	213	3300	4.7	-	9462	6	83
Fjeldtvedt et al. [48]	ZYNQ7020	147	2350	0.3	-	5872	6	84
Orlandic et al. [49]	ZYNQ7020	150	12000	0.5	-	14709	7	37
This Work	ZYNQ7020	220	6259	3.7	2	15188	220	135
This Work	ZYNQ7020	220	14083	3.7	4.7	15188	220	135
This Work	ZYNQ7020	220	18778	2.6	14	10253	112	107

embedded GPU and 216 times faster than the solution with the dual-core ARM processor.

Considering power and energy consumption, the proposed solution is also better (see energy results in Table III). However, it should be noted that the GPU and ARM implementations were not subject to the same development effort. So, their results can potentially be improved, reducing the gap to the FPGA solution.

Since the SoC FPGA needs less power and executes the algorithm faster, the energy is from 77 to 119 times lower than the energy used by the embedded GPU and 146 to 99 times lower than the energy of the ARM processor.

C. Comparison With Other FPGA-Based Platforms

As far as we know, there is no other previous implementation of compressive sensing in FPGA. Most of the related work on FPGA for compressive sensing methods are concerned with the reconstruction part, which can be computed on the ground-based station. However, there are a few developed designs on FPGA for CCSDS 123 recommendation used to compress hyperspectral images. A comparison of our work against implementations of recommendation CCSDS 123 has been made (see Table IV).

The reported resources and power of the proposed architecture are for the whole system, including the DDI module. So, it reports in general more resources. The power in [48] and [49] only refers to the dedicated hardware block and does not include the power of the processor and data transfer. The proposed architecture can achieve a compression ratio of 14 times, whereas the reported CCSDS implementations have a compression factor lower than 4.6. The highest throughputs are also achieved with the proposed architecture. With a compression rate of 14, the throughput is 56% better than the throughput reported in [49].

Another aspect of the previous architectures is that it is not clear how data are sent and received from the main computing core. A large percentage of the area and consumed energy of the proposed architecture is relative to hardware for data communication.

Also, the results of the proposed architecture are for the highest throughput. However, as shown previously, the required resources reduce drastically if the main goal is only to achieve enough performance for real-time performance.

D. Comparison With Other Methods

The performance of the proposed SoC architecture is compared with other methods, namely the compressive sensing method called SPECA, which was introduced in [28] and with a parallel version of HYCA introduced in [27].

TABLE V
COMPARISON OF THE EXECUTION TIMES (IN SECONDS) FOR THE COMPRESSIVE SENSING ALGORITHMS, FOR A COMPRESSION RATE OF 14.6 CONSIDERING AN IMAGE WITH 512 LINES, 614 SAMPLES, AND 224 BANDS

Work	device	Time
This work	ZYNQ7020	0.07
[27]	Intel i7-4770K	5.99
[27]	GeForce GTX 590	0.135
[27]	GeForce GTX TITAN	0.107

TABLE VI
COMPARISON OF THE EXECUTION TIMES (IN SECONDS) FOR THE COMPRESSIVE SENSING ALGORITHMS, FOR A COMPRESSION RATE OF 7.7 CONSIDERING AN IMAGE WITH 512 LINES, 614 SAMPLES, AND 224 BANDS

Work	Device	Time
This work	ZYNQ7020	0.09
[28]	Intel i7-4790	3.07
[28]	GeForce GTX 980	0.32

In Tables V and VI, the methods execution time is compared. In [27], the method is tested on three different platforms. First, an Intel i7-4770 K CPU at 3.50 GHz with four physical cores and 32 GB of DDR3 RAM memory; second, a GPU on a NVIDIA GeForce GTX 590 board, which features 1024 processor cores operating at 1.215 GHz, total dedicated memory of 3072 MB; and finally; a GPU on a NVIDIA GeForce GTX TITAN board, which features 2688 processor cores operating at 876 MHz, total dedicated memory of 6144 MB. In [28], SPECA is tested on a Intel i7-4790 CPU at 3.6-GHz clock speed connected to 32-GB RAM memory and on a GPU NVIDIA GeForce GTX 980, which contains 16 multiprocessors with 128 CUDA cores each at 1.33-GHz and 4-GB of memory. From the results presented, one can conclude that, for the same compression ratio, the execution time of the proposed work is lower than others methods.

The accuracy of the proposed SoC architecture is compared with the lossy compression method introduced in [50]. This method employs a prediction-based scheme, with quantization and rate-distortion optimization, with a low complexity technique in terms of memory and computational requirements.

The experiments are carried out using the Yellostone AVIRIS image labeled as "Scene0" from flight f060925t01p00r12. Fig. 9 presents the accuracy in terms of the peak signal-to-noise ratio

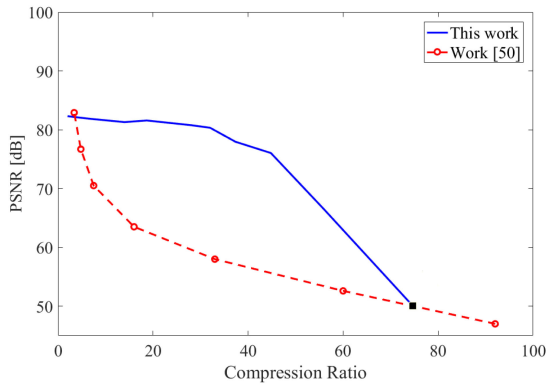


Fig. 9. PSNR in decibels for different compression ratios. Proposed architecture (solid line) and [50] (dashed line).

(PSNR) for different compression ratios. For the proposed architecture, the image reconstruction is done with the P-HYCA reconstruction algorithm [27] in a desktop computer. On this figure, the proposed architecture has a PSNR higher than 80 dB for a compression ratio smaller than 30×. The PSNR start dropping for a compression ratio higher than 50×. The reported results for [50] are better in terms of PSNR for a compression ratio higher than 75×.

V. Conclusions

On-board processing systems have recently emerged in order to overcome the huge amount of data to transfer from the satellite to the ground station. Hyperspectral imagery is a remote sensing technology that can benefit from on-board processing. This article proposes a hardware/software FPGA-based architecture for compressive sensing of hyperspectral images.

The original algorithm was reorganized to improve the accesses to data stored in the external memory. The proposed architecture has been designed in a Xilinx Zynq board with a Zynq-7020 SoC FPGA. Experimental results with real hyperspectral datasets indicate that the proposed implementation can fulfill real-time requirements with low resources in a low-cost SoC FPGA. The architecture is also around 100 times more energy efficient when compared to a software only solution and to an embedded GPU.

Since the FPGA permits to configure the architecture for other custom bitwidths, the algorithm and the architecture can be further optimized for specific bitwidths will less than 16 bits. However, it requires an additional analysis to determine how the bitwidth reduction influences the signal-to-noise ratio.

References

[1] T. Adão *et al.*, "Hyperspectral imaging: A review on UAV-based sensors, data processing and applications for agriculture and forestry," *Remote Sens.*, vol. 9, no. 11, 2017. [Online]. Available: https://www.mdpi.com/2072-4292/9/11/1110

[2] S. B. Tombet, F. Marcotte, É. Guyot, M. Chamberland, and V. Farley, "Toward UAV based compact thermal infrared hyperspectral imaging solution for real-time gas detection identification and quantification (Conference Presentation)," *Proc. SPIE*, 2019. [Online]. Available: https://doi.org/10.1117/12.2521191

[3] P. W. Yuen and M. Richardson, "An introduction to hyperspectral imaging and its application for security, surveillance and target acquisition," *Imag. Sci. J.*, vol. 58, no. 5, pp. 241–253, 2010. [Online]. Available: https://doi.org/10.1179/174313110X12771950995716

[4] H. Fabelo *et al.*, "A novel use of hyperspectral images for human brain cancer detection using in-vivo samples," in *Proc. 9th Int. Joint Conf. Biomed. Eng. Syst. Technol.*, 2016, pp. 311–320.

[5] X. Fu and J. Chen, "A review of hyperspectral imaging for chicken meat safety and quality evaluation: Application, hardware, and software," *Comprehensive Rev. Food Sci. Food Saf.*, vol. 18, no. 2, pp. 535–547, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/1541-4337.12428

[6] Y.-Z. Feng and D.-W. Sun, "Application of hyperspectral imaging in food safety inspection and control: A review," *Crit. Rev. Food Sci. Nutri.*, vol. 52, no. 11, pp. 1039–1058, 2012. [Online]. Available: https://doi.org/10.1080/10408398.2011.651542

[7] J. Yang, D. W. Messinger, and R. R. Dube, "Bloodstain detection and discrimination impacted by spectral shift when using an interference filter-based visible and near-infrared multispectral crime scene imaging system," *Opt. Eng.*, vol. 57, no. 3, pp. 1–10, 2018. [Online]. Available: https://doi.org/10.1117/1.OE.57.3.033101

[8] B. Li, P. Beveridge, W. T. O'Hare, and M. Islam, "The application of visible wavelength reflectance hyperspectral imaging for the detection and identification of blood stains," *Sci. Justice*, vol. 54, no. 6, pp. 432–438, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1355030614000586

[9] P. Ghamisi *et al.*, "Advances in hyperspectral image and signal processing: A comprehensive overview of the state of the art," *IEEE Geosci. Remote Sens. Mag.*, vol. 5, no. 4, pp. 37–78, Dec. 2017.

[10] X. Ceamanos and S. Valero, "Processing hyperspectral images," in *Opt. Rem. Sens. of Land Sur.*, N. Baghdadi and M. Zribi, Eds. New York, NY, USA: Elsevier, 2016, pp. 163–200. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9781785481024500041

[11] R. Guerra, Y. Barrios, M. Díaz, L. Santos, S. López, and R. Sarmiento, "A new algorithm for the on-board compression of hyperspectral images," *Remote Sens.*, vol. 10, no. 3, p. 428, 2018, doi: 10.3390/rs10030428.

[12] D. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, Apr. 2006.

[13] E. J. Candes and M. B. Wakin, "An introduction to compressive sampling," *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 21–30, Mar. 2008.

[14] Y. Wu, M. Rosca, and T. Lillicrap, "Deep compressed sensing," in *Proc. 36th Int. Conf. Mach. Learn.*, Jun. 9–15, 2019, pp. 6850–6860. [Online]. Available: http://proceedings.mlr.press/v97/wu19d.html

[15] M. F. Duarte and Y. C. Eldar, "Structured compressed sensing: From theory to applications," *IEEE Trans. Signal Process.*, vol. 59, no. 9, pp. 4053–4085, Sep. 2011.

[16] R. M. Willett, M. F. Duarte, M. A. Davenport, and R. G. Baraniuk, "Sparsity and structure in hyperspectral imaging: Sensing, reconstruction, and target detection," *IEEE Signal Process. Mag.*, vol. 31, no. 1, pp. 116–126, Jan. 2014.

[17] J. M. Bioucas-Dias and J. M. P. Nascimento, "Hyperspectral subspace identification," *IEEE Trans. Geosci. Remote Sens.*, vol. 46, no. 8, pp. 2435–2445, Aug. 2008.

[18] 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.

[19] C. R. Berger, Z. Wang, J. Huang, and S. Zhou, "Application of compressive sensing to sparse channel estimation," *IEEE Commun. Mag.*, vol. 48, no. 11, pp. 164–174, Nov. 2010.

[20] Y. Oiknine, I. August, V. Farber, D. Gedalin, and A. Stern, "Compressive sensing hyperspectral imaging by spectral multiplexing with liquid crystal," *J. Imag.*, vol. 5, no. 1, p. 3, 2019, doi: 10.3390/jimaging5010003.

[21] P. Xu, B. Chen, L. Xue, J. Zhang, and L. Zhu, "A prediction-based spatial-spectral adaptive hyperspectral compressive sensing algorithm," *Sensors*, vol. 18, no. 10, p. 3289, 2018, doi: 10.3390/s18103289.

[22] G. Martin, J. M. Bioucas-Dias, and A. Plaza, "HYCA: A new technique for hyperspectral compressive sensing," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 5, pp. 2819–2831, May 2015.

[23] Z. Xiong, Z. Shi, H. Li, L. Wang, D. Liu, and F. Wu, "HSCNN: CNN-based hyperspectral image recovery from spectrally undersampled projections," in *Proc. IEEE Int. Conf. Comput. Vis. Workshops*, 2017, pp. 518–525.

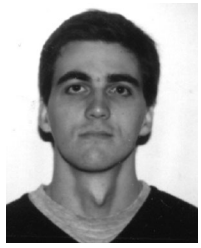
[24] I. Choi, D. S. Jeon, G. Nam, D. Gutierrez, and M. H. Kim, "High-quality hyperspectral reconstruction using a spectral prior," *ACM Trans. Graph.*, vol. 36, no. 6, pp. 218-1–218-13, Nov. 2017. [Online]. Available: http://doi.acm.org/10.1145/3130800.3130810

[25] L. Zhang, W. Wei, Y. Zhang, C. Shen, A. van den Hengel, and Q. Shi, "Dictionary learning for promoting structured sparsity in hyperspectral compressive sensing," *IEEE Trans. Geosci. Remote Sens.*, vol. 54, no. 12, pp. 7223–7235, Dec. 2016.
[26] G. Martín and J. M. Bioucas-Dias, "Hyperspectral blind reconstruction from random spectral projections," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 6, pp. 2390–2399, Jun. 2016.
[27] S. Bernabe, G. Martin, J. Nascimento, J. Bioucas-Dias, A. Plaza, and V. Silva, "Parallel hyperspectral coded aperture for compressive sensing on GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 932–944, Feb. 2016.
[28] J. Sevilla, G. Martin, J. Nascimento, and J. Bioucas-Dias, "Hyperspectral image reconstruction from random projections on GPU," in *Proc. IEEE Int. Geosci. Remote Sens. Symp.*, 2016, pp. 280–283.
[29] J. Sevilla, G. Martin, and J. M. Nascimento, "Parallel hyperspectral image reconstruction using random projections," *Proc. SPIE*, vol. 10007, pp. 1 000 707-1–1 000 707–9, 2016.
[30] J. M. Haut, S. Bernabé, M. E. Paoletti, R. Fernandez-Beltran, A. Plaza, and J. Plaza, "Low-high-power consumption architectures for deep-learning models applied to hyperspectral image classification," *IEEE Geosci. Remote Sens. Lett.*, vol. 16, no. 5, pp. 776–780, May 2019.
[31] M. Díaz *et al.*, "Real-time hyperspectral image compression onto embedded GPUs," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 12, no. 8, pp. 2792–2809, Aug. 2019.
[32] A. Rodríguez, L. Santos, R. Sarmiento, and E. D. L. Torre, "Scalable hardware-based on-board processing for run-time adaptive lossless hyperspectral compression," *IEEE Access*, vol. 7, pp. 10 644–10 652, 2019.
[33] D. Valsesia and E. Magli, "High-throughput onboard hyperspectral image compression with ground-based CNN reconstruction," *IEEE Trans. Geosci. Remote Sens.*, vol. 57, no. 12, pp. 9544–9553, Dec. 2019.
[34] D. Báscones, C. González, and D. Mozos, "FPGA implementation of the CCSDS 1.2.3 standard for real-time hyperspectral lossless compression," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 4, pp. 1158–1165, Apr. 2018.
[35] J. Fjeldtvedt, M. Orlandić, and T. A. Johansen, "An efficient real-time FPGA implementation of the CCSDS-123 compression standard for hyperspectral images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 10, pp. 3841–3852, Oct. 2018.
[36] A. Plaza, Q. Du, Y.-L. Chang, and R. King, "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 4, no. 3, pp. 528–544, Sep. 2011.
[37] H. Quinn, "Radiation effects in reconfigurable FPGAs," *Semicond. Sci. Technol.*, vol. 32, no. 4, Mar. 2017, Art. no. 044001.
[38] M. Wirthlin, "High-reliability FPGA-based systems: Space, high-energy physics, and beyond," *Proc. IEEE*, vol. 103, no. 3, pp. 379–389, Mar. 2015.
[39] S. Lopez, T. Vladimirova, C. Gonzalez, J. Resano, D. Mozos, and A. Plaza, "The promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends," *Proc. IEEE*, vol. 101, no. 3, pp. 698–722, Mar. 2013.
[40] J. Eckstein and D. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Math. Program.*, vol. 5, pp. 293–318, 1992.
[41] G. Martín, J. M. Bioucas-Dias, and A. Plaza, "HYCA: A new technique for hyperspectral compressive sensing," *IEEE Trans. Geosci. Remote Sens.*, vol. 53, no. 5, pp. 2819–2831, May 2015.
[42] A. B. Kiely and M. A. Klimesh, "Exploiting calibration-induced artifacts in lossless compression of hyperspectral imagery," *IEEE Trans. Geosci. Remote Sens.*, vol. 47, no. 8, pp. 2672–2678, Aug. 2009.
[43] R. Green *et al.*, "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sens. Environ.*, vol. 65, no. 3, pp. 227–248, 1998.
[44] J. M. P. Nascimento and M. Véstias, "Hyperspectral compressive sensing: A comparison of embedded GPU and ARM implementations," *Proc. SPIE*, vol. 11163, pp. 88–97, 2019. [Online]. Available: https://doi.org/10.1117/12.2532581
[45] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, "GPU scheduling on the NVIDIA TX2: Hidden details revealed," in *Proc. IEEE Real-Time Syst. Symp.*, Dec. 2017, pp. 104–115.
[46] L. Santos, L. Berrojo, J. Moreno, J. F. López, and R. Sarmiento, "Multispectral and hyperspectral lossless compressor for space applications (HYLOC): A low-complexity FPGA implementation of the CCSDS 123 Standard," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 9, no. 2, pp. 757–770, Feb. 2016.
[47] A. Tsigkanos, N. Kranitis, G. A. Theodorou, and A. Paschalis, "A 3.3 GBPs CCSDS 123.0-B-1 multispectral hyperspectral image compression hardware accelerator on a space-grade SRAM FPGA," *IEEE Trans. Emerg. Topics Comput.*, to be published, doi: 10.1109/TETC.2018.2854412.
[48] J. Fjeldtvedt, M. Orlandić, and T. A. Johansen, "An efficient real-time FPGA implementation of the CCSDS-123 compression standard for hyperspectral images," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 11, no. 10, pp. 3841–3852, Oct. 2018.
[49] M. Orlandic, J. Fjeldtvedt, and T. A. Johansen, "A parallel FPGA implementation of the CCSDS-123 compression algorithm," *Remote Sens.*, vol. 11, p. 673, 2019, doi: 10.3390/rs11060673.
[50] A. Abrardo, M. Barni, and E. Magli, "Low-complexity predictive lossy compression of hyperspectral and ultraspectral images," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2011, pp. 797–800.

José M. P. Nascimento received the Ph.D. degree in electrical and computer engineering from the Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, in 2006.

He is currently a Professor with Instituto Superior de Engenharia de Lisboa, Lisbon, and a Researcher with Instituto de Telecomunicações, Lisbon. He has contributed to more than 60 journal, international conference papers, and book chapters. His current research interests include remote sensing, image processing, and high-performance computing.

Dr. Nascimento is currently serving as a Reviewer of several international journals and he has also been a member of Program/Technical Committees of several international conferences.



Mário P. Véstias received the Ph.D. degree in electrical and computer engineering from the Technical University of Lisbon, Lisbon, Portugal, in 2002.

He is a Coordinate Professor with the Department of Electronic, Telecommunications and Computer Engineering, School of Engineering, Polytechnic Institute of Lisbon, Lisbon. He is a Senior Researcher with the Electronic Systems Design and Automation group, Instituto de Engenharia de Sistemas e Computadores-Investigação e Desenvolvimento, Lisbon. His main research interests include computer architectures and digital systems for high-performance embedded computing, with an emphasis on reconfigurable computing.

Gabriel Martín received the Ph.D. degree in computer engineering from the University of Extremadura, Caceres, Spain, in 2013.

He was a Predoctoral Research Associate (funded by the Spanish Ministry of Science and Innovation) with the Hyperspectral Computing Laboratory and Postdoctoral Researcher with "Instituto de Telecomunicações," Lisbon, Portugal. He is currently working as a Senior Performance Analytics Engineer with Atrio Inc., CA, USA. He is coauthor of a patent about a portable performance analytics system. He has authored or co-authored more than 60 publications, including several book chapters, journal citation report papers, and peer-reviewed international conference papers. His research interest include hyperspectral image processing, specifically the areas of unmixing and compressive sensing of hyperspectral images, as well as the efficient processing of these images in high-performance computing architectures such as graphics processing units.

Dr. Martín has served as a Reviewer for the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING and the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING. He was the recipient of several awards for his Ph.D. dissertation such as the "Best Iberian Ph.D. Dissertation in Information System and Technologies" awarded by the Iberian Association for Information Systems and Technologies and the "Outstanding Ph.D. Dissertation award" by the University of Extremadura.