FPGA Implementation of Endmember Extraction Algorithms from Hyperspectral Imagery: Pixel Purity Index versus N-FINDR

Carlos González[a], Daniel Mozos[a], Javier Resano[b] and Antonio Plaza[c]

[a]Department of Computer Architecture and Automatics, Complutense University of Madrid, C/ Profesor José García Santesmases s/n 28040 Madrid (Spain)
[b]Department of Computer Architecture, University of Zaragoza, C/ de María de Luna 3, 50018 Zaragoza (Spain)
[c]Department of Technology of Computers and Communications, University of Extremadura, Avda. de la Universidad s/n E-10071 Cáceres (Spain)

ABSTRACT

Endmember extraction is an important task for remotely sensed hyperspectral data exploitation. It comprises the identification of spectral signatures corresponding to macroscopically pure components in the scene, so that mixed pixels (resulting from limited spatial resolution, mixing phenomena happening at different scales, etc.) can be decomposed into combinations of pure component spectra weighted by an estimation of the proportion (abundance) of each endmember in the pixel. Over the last years, several algorithms have been proposed for automatic extraction of endmembers from hyperspectral images. These algorithms can be time-consuming (particularly for high-dimensional hyperspectral images). Parallel computing architectures have offered an attractive solution for fast endmember extraction from hyperspectral data sets, but these systems are expensive and difficult to adapt to on-board data processing scenarios, in which low-weight and low-power hardware components are essential to reduce mission payload, overcome downlink bandwidth limitations in the transmission of the hyperspectral data to ground stations on Earth, and obtain analysis results in (near) real-time.

In this paper, we perform an inter-comparison of the hardware implementations of two widely used techniques for automatic endmember extraction from remotely sensed hyperspectral images: the pixel purity index (PPI) and the N-FINDR. The hardware versions have been developed in field programmable gate arrays (FPGAs). Our study reveals that these reconfigurable hardware devices can bridge the gap towards on-board processing of remotely sensed hyperspectral data and provide implementations that can significantly outperform the (optimized) equivalent software versions of the considered endmember extraction algorithms.

Keywords: Hyperspectral image analysis, endmember extraction, pixel purity index (PPI), N-FINDR, field programmable gate arrays (FPGAs).

1. INTRODUCTION

Hyperspectral imaging, also known as imaging spectroscopy, is a technique that has been widely used during recent years in Earth and planetary remote sensing.[1] It generates hundreds of images, corresponding to different wavelength channels, for the same area on the surface of the Earth. The concept of hyperspectral imaging originated at NASA's Jet Propulsion Laboratory in California, which developed instruments such as the Airborne Imaging Spectrometer (AIS), then called AVIRIS (for Airborne Visible Infra-Red Imaging Spectrometer[2]). This system is now able to cover the wavelength region from 400 to 2500 nanometers using 224 spectral channels, at nominal spectral resolution of 10 nanometers. As a result, each pixel (considered as a vector) collected by a hyperspectral instrument can be seen as a *spectral signature* or 'fingerprint' of the underlying materials within the pixel (see Figure 1).

Further author information: (Send correspondence to Carlos González)
Carlos González: carlosgonzalez@fdi.ucm.es, Daniel Mozos: mozos@fis.ucm.es, Javier Resano: jresano@unizar.es, Antonio Plaza: aplaza@unex.es

High-Performance Computing in Remote Sensing, edited by Bormin Huang, Antonio J. Plaza,
Proc. of SPIE Vol. 8183, 81830F · © 2011 SPIE · CCC code: 0277-786X/11/$18 · doi: 10.1117/12.897384

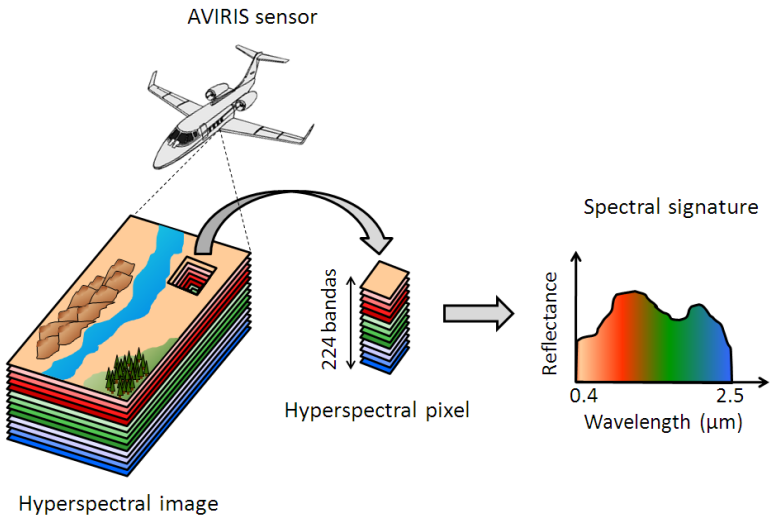


Figure 1. The concept of hyperspectral imaging.

One of the great challenges in remotely sensed hyperspectral image analysis is computational complexity resulting from the need to process enormous data volumes.[3] With recent advances in reconfigurable computing, many image processing algorithms can be accelerated using high-performance FPGAs.[4] One of the fundamental tasks in hyperspectral image processing is endmember extraction which has found many applications in data exploitation, especially spectral unmixing.[5] Over the last years, many algorithms have been developed with the purpose of finding "spectral endmembers",[5] which are assumed to be pure signatures in hyperspectral data sets. Such pure signatures can then be used to estimate the abundance or concentration of materials in mixed pixels, thus allowing sub-pixel analysis of hyperspectral images. The pixel purity index (PPI) and the N-FINDR algorithms have been widely used in endmember extraction. These algorithms have a very expensive computational cost, a fact that has generally prevented its exploitation in valid response times in a wide range of applications, including environmental monitoring, military applications or hazard and threat assessment/tracking. The flexibility, high performance and reduced energy consumption of FPGAs make them particularly attractive in remote sensing applications which require a response in real- or near real-time.[4]

The remainder of the paper is organized as follows. Section 2 formulates the spectral unmixing problem in mathematical terms. Section 3 describes the original PPI and N-FINDR algorithms. Section 4 describes their implementations on a Xilinx Virtex-4 XC4VFX60 FPGA. Section 5 provides an experimental assessment of both endmember extraction accuracy and processing performance of the proposed FPGA-based algorithms, using well-known hyperspectral data sets collected by AVIRIS over two different sites: the Cuprite mining district in Nevada, and the Jasper Ridge Biological Preserve in California. Finally, section 6 concludes with some remarks and hints at plausible future research lines.

2. SPECTRAL UNMIXING

In order to define the spectral unmixing problem in mathematical terms, let us assume that a remotely sensed hyperspectral scene with n bands is denoted by $\mathbf{F}$, in which the pixel at the discrete spatial coordinates (i, j) of the scene is represented by a vector $\mathbf{X}(i, j) = [x_1(i, j), x_2(i, j), \cdots, x_n(i, j)] \in \Re^n$, where $\Re$ 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[3],[6] each pixel vector in the original scene can be modeled using the following expression:

$$\mathbf{X}(i,j) = \sum_{z=1}^{p} \Phi_z(i,j) \cdot \mathbf{E}_z + \mathbf{n}(i,j), \tag{1}$$

where $\mathbf{E}_z$ denotes the spectral response of endmember z, $\Phi_z(i,j)$ is a scalar value designating the fractional abundance of the endmember z at the pixel $\mathbf{X}(i,j)$, p is the total number of endmembers, and $\mathbf{n}(i,j)$ is a noise vector. The solution of the linear spectral mixture problem described in (1) relies on the correct determination of a set $\{\mathbf{E}_z\}_{z=1}^{p}$ of endmembers and their correspondent abundance fractions $\{\Phi_z(i,j)\}_{z=1}^{p}$ at each pixel $\mathbf{X}(i,j)$. The derivation and validation of the correct suite of endmembers has remained a challenging and goal for the past years (not only in terms of adequate spectral signature extraction,[5] but also in terms of computational complexity[7]).

3. ALGORITHMS DESCRIPTION

3.1 The Pixel Purity Index (PPI) Algorithm

The PPI algorithm calculates a spectral purity score for each n-dimensional pixel in the original data by generating random unit vectors (called *skewers*), so that all pixel vectors are projected onto the skewers and the ones falling at the extremes of each skewer are counted. After many repeated projections to different skewers, those pixels that count above a certain cut-off threshold are declared "pure".

The inputs to the PPI algorithm are a hyperspectral image cube $\mathbf{F}$ with N spectral bands; a maximum number of projections, K; a cut-off threshold value, v_c, used to select as final endmembers only those pixels that have been selected as extreme pixels at least v_c times throughout the process; and a threshold angle, v_a, used to discard redundant endmembers during the process. The output is a set of p endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$. The algorithm can be summarized by the following steps:

1. *Skewer generation.* Produce a set of K randomly generated unit vectors, denoted by $\{\mathbf{skewer}_j\}_{j=1}^{K}$.
2. *Extreme projections.* For each $\mathbf{skewer}_j$, all sample pixel vectors $\mathbf{f}_i$ in the original data set $\mathbf{F}$ are projected onto $\mathbf{skewer}_j$ via dot products of $\mathbf{f}_i \cdot \mathbf{skewer}_j$ to find sample vectors at its extreme (maximum and minimum) projections, forming an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}(\mathbf{skewer}_j)$.
3. *Calculation of pixel purity scores.* Define an indicator function of a set S, denoted by $I_S(\mathbf{f}_i)$, to denote membership of an element $\mathbf{f}_i$ to that particular set as $I_S(\mathbf{f}_i) = 1$ if $(\mathbf{f}_i \in S)$ else 0. Using the function above, calculate the number of times that a given pixel has been selected as extreme using the following equation:

$$N_{PPI}(\mathbf{f}_i) = \sum_{j=1}^{K} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i) \tag{2}$$

4. *Endmember selection.* Find the pixels with value of $N_{PPI}(\mathbf{f}_i)$ above v_c and form a unique set of p endmembers $\{\mathbf{e}_j\}_{j=1}^{p}$ by calculating the spectral angle (SA)[3,8] for all possible endmember pairs and discarding those which result in an angle value below v_a. The SA is invariant to multiplicative scalings that may arise due to differences in illumination and sensor observation angle.[9]

The most time consuming stage of the PPI algorithm is given by step 2 (extreme projections). Fortunately, the PPI algorithm is well suited for parallel implementation. The computation of skewer projections are independent and can be performed simultaneously, leading to many ways of parallelization.

3.2 The N-FINDR Algorithm

This algorithm attempts to automatically find the simplex of maximum volume that can be inscribed within the hyperspectral data set. The original N-FINDR algorithm developed by Winter[10] can be summarized as follows:

1. *Feature reduction.* Apply a dimensionality reduction transformation such as the minimum noise fraction (MNF)[11] or the principal component analysis (PCA)[12] 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 $\{\mathbf{E}_1^{(0)}, \mathbf{E}_2^{(0)}, \cdots, \mathbf{E}_p^{(0)}\}$ be a set of endmembers randomly extracted from the input data.

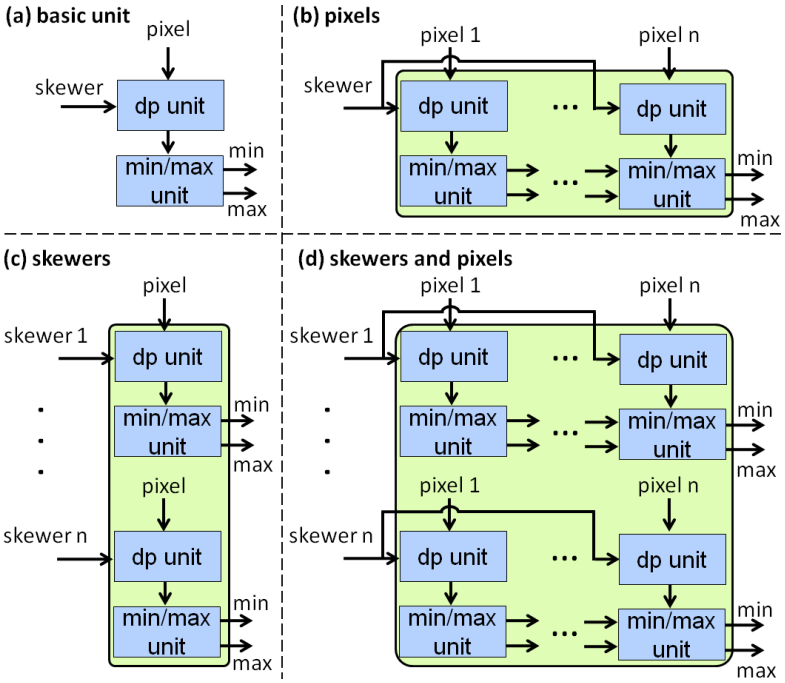


Figure 2. (a) basic unit. (b) Parallelization strategy by pixels. (c) Parallelization strategy by skewers. (d) Parallelization strategy by skewers and pixels.

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

$$V(\mathbf{E}_1^{(k)}, \mathbf{E}_2^{(k)}, \cdots, \mathbf{E}_p^{(k)}) = \frac{\left| det \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \mathbf{E}_1^{(k)} & \mathbf{E}_2^{(k)} & \cdots & \mathbf{E}_p^{(k)} \end{bmatrix} \right|}{(p-1)!}, \tag{3}$$

4. *Replacement.* For each pixel vector $\mathbf{X}(i,j)$ in the input hyperspectral data, recalculate the volume by testing the pixel in all p endmember positions, i.e., first calculate $V(\mathbf{X}(i,j), \mathbf{E}_2^{(k)}, \cdots, \mathbf{E}_p^{(k)})$, then $V(\mathbf{E}_1^{(k)}, \mathbf{X}(i,j), \cdots, \mathbf{E}_p^{(k)})$, and so on, until $V(\mathbf{E}_1^{(k)}, \mathbf{E}_2^{(k)}, \cdots, \mathbf{X}(i,j))$. If none of the p recalculated volumes is greater than $V(\mathbf{E}_1^{(k)}, \mathbf{E}_2^{(k)}, \cdots, \mathbf{E}_p^{(k)})$, 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 $\mathbf{E}_j^{(k+1)}$. In this case, a new set of endmembers is produced by letting $\mathbf{E}_j^{(k+1)} = \mathbf{X}(i,j)$ and $\mathbf{E}_i^{(k+1)} = \mathbf{E}_i^{(k)}$ for $i \neq j$. The replacement step is repeated in an iterative fashion, using as many iterations as needed until there are no more replacements of endmembers.

In this work, we use the PCA of ENVI 4.0 software to generate dimensionally reduced images to be the input of the N-FINDR algorithm.

4. FPGA IMPLEMENTATIONS

4.1 The Pixel Purity Index (PPI) Algorithm

The most time consuming stage of the PPI algorithm (extreme projections) computes a very large number of dot products, all of which can be performed simultaneously. If we consider a simple basic unit such as the one displayed in Fig. 2(a) as the baseline for parallel computations, then we can perform the parallel computations by pixels [see Fig. 2(b)], by skewers [see Fig. 2(c)], or by pixels and skewers [see Fig. 2(d)]. If we parallelize the computations by skewers, we can compute K dot products at the same time for the same pixel, where K

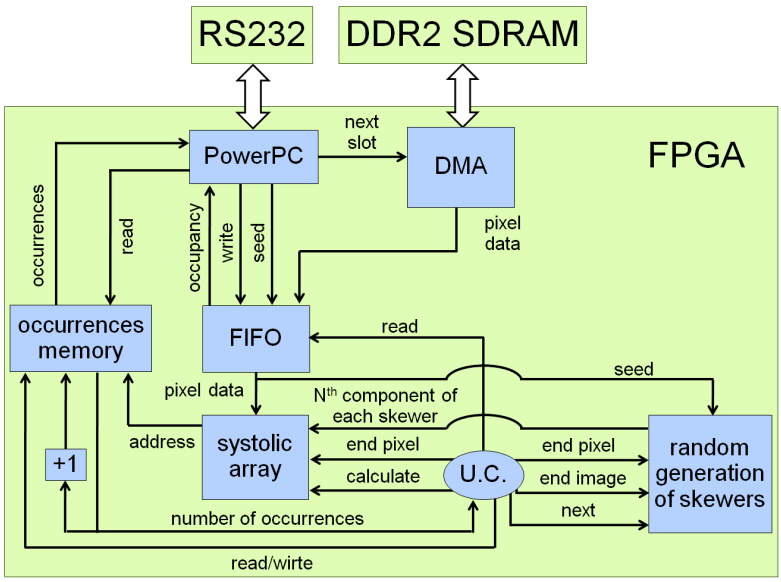


Figure 3. Hardware architecture to implement the endmember extraction step.

is the number of skewers [see Fig. 2(c)]. If we increase the number of skewers, the required area would grow proportionally with the number of dot product units and the clock cycle would remain constant. Another possible way to parallelize the extreme projections stage is to compute them by pixels. In this implementation, the increase in the number of pixels makes the required area and the clock cycle will be equal than the parallelization by skewers. However, as we increase the number of parallel computations, a greater number of additional clock cycles would be needed for maxima/minima computations. Finally, the parallelization strategy in Fig. 2(d) is an intermediate solution which provides no further advantage with respect to the parallelization by skewers and has the same extra clock cycles that parallelization by pixels.

Taking in mind the above rationale, in this work we have selected the parallelization strategy based on skewers. Apart from the aforementioned advantages with regards to other possible strategies, another reason for our selection is that the parallelization strategy based on skewers fits very well the procedure for data collection (in a pixel-by-pixel fashion) at the imaging instrument. Therefore, parallelization by skewers is the one that best fits the data entry mechanism since each pixel can be processed immediately as collected. Specifically, our hardware system should be able to compute K dot products against the same pixel $\mathbf{f}_i$ at the same time, being K the number of skewers.

Fig. 3 shows the architecture of the hardware used to implement the PPI algorithm, along with the I/O communications. For data input, we use a DDR2 SDRAM and a DMA (controlled by a PowerPC) with a FIFO to store pixel data. For data output, we use a PowerPC to send the position of the endmembers via a RS232 port. Finally, a systolic array, a random generation module and a occurrences memory are also used. For illustrative purposes, Fig. 4 describes the architecture of the dot product processors used in our systolic array. Basically, a systolic cycle consists of computing a single dot product between a pixel and a skewer, and to memorize the index of the pixel if the dot product is higher or smaller than a previously computed Max/Min value. It has been shown in previous work[13,14] that the skewer values can be limited to a very small set of integers when their dimensionality is large, as in the case of hyperspectral images. A particular and interesting set is $\{1, -1\}$ since it avoids the multiplication. The dot product is thus reduced to an accumulation of positive and negative values. As a result, each dot product processor only needs to accumulate the positive or negative values of the pixel input according to the skewer input. These units are thus only composed of a single addition/subtraction operator and a register. The min/max unit receives the result of the dot product and compares it with the previous minimum and maximum values. If the result is a new minimum or maximum, it will be stored for future comparisons.

On the other hand, the incorporation of a hardware-based random generation module is one of the main features of our system. This module significantly reduces the I/O communications that were the main bottleneck

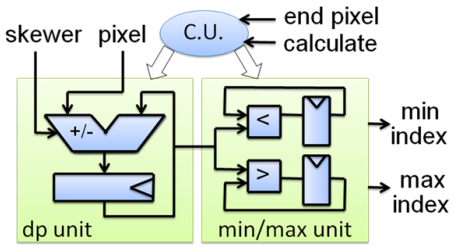


Figure 4. Hardware architecture of a dot product processor.

Table 1. Summary of Resource Utilization for the FPGA-Based Implementation of the PPI Algorithm on a Virtex-4 XC4VFX60 FPGA.

Component	Number of skewers	Number of slice flip flops	Number of 4 input LUTs	Number of slices	Percentage of total	Maximum frequency (MHz)
Systolic Array	120	13440	23660	12711	53.16	217
	140	15680	27603	14830	58.66	217
	160	17920	31546	16948	67.04	217
	180	20160	35489	19067	75.42	217
	200	22400	39432	21148	83.65	217
Random Generation Module	120	240	720	346	1.36	684
	140	280	840	403	1.59	684
	160	320	960	461	1.82	684
	180	360	1080	521	2.06	684
	200	400	1200	576	2.27	684
RS232 Transmitter	-	69	128	71	0.28	238
DMA Controller	-	170	531	367	1.45	102

of the system in previous implementations.[13–15] It should be noted that, in a digital system, it is not possible to generate 100% random numbers. In our design, we have implemented a random generator module which provides pseudo-random and uniformly-distributed sequences using registers and XOR gates. It requires an affordable amount of space (576 slices for 200 skewers) and it is able to generate the next component of every skewer in only one clock cycle and operates at a high clock frequency (648 MHz).

To calculate the number of times each pixel has been selected as extreme (step 3 of the algorithm) we use the occurrences memory, which is initialized to zero. Once we have calculated the minimum and maximum value for each of the skewers, we update the number of occurrences by reading the previous values stored for the extremes in the occurrences memory and then, writing these values increased by one. When this step is completed, the PowerPC reads the total number of occurrences for each pixel. If this number exceeds the threshold value v_c, it is selected as an endmember. After that, the PowerPC calculates the SA for all possible endmember pairs and discards those which result in an angle value below v_a (step 4 of the algorithm). Finally, the PowerPC sends the non-redundant endmember positions through the RS232 port.

Table 1 shows the resources used for our FPGA-based implementation of the PPI endmember extraction process for different numbers of skewers (ranging from $K = 140$ to $K = 200$), tested on the Virtex-4 XC4VFX60 FPGA of the ML410 board. As shown by Table 1, we can scale our design up to 200 skewers (therefore, $P = 50$ algorithm passes are needed in order to process $K = 10^4$ skewers). An interesting feature of our design is that we can scale it without increasing the delay of the critical path.

4.2 The N-FINDR Algorithm

Figure 5 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 FIFO to store pixel data. N-FINDR module is used to implement our version of the N-FINDR algorithm. Finally, a transmitter is used to send the endmembers via a RS232 port.

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, makes it difficult to develop an efficient

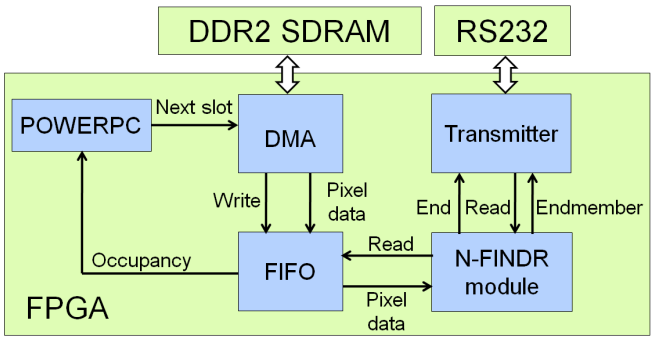


Figure 5. Hardware architecture of the complete system.

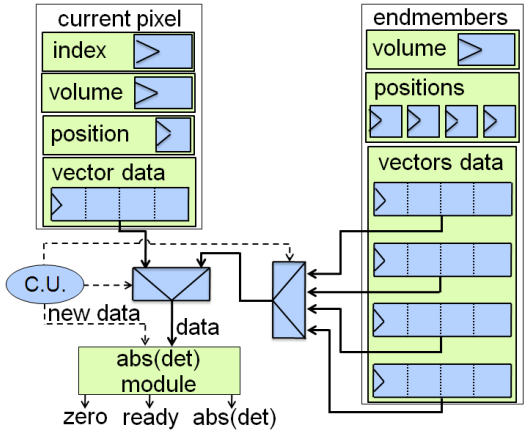


Figure 6. Hardware architecture to implement the N-FINDR algorithm.

implementation of the algorithm and this is the reason because there are not FPGA implementations of the algorithm in the literature. To calculate determinants, it is advisable to use the fundamental properties of the determinants and apply them systematically to transform the determinant in others who are increasingly easy to calculate, down to one which is trivial. For the design of the algorithm we use the Gauss elimination method in order to have a triangular matrix.

Figure 6 shows the hardware architecture used to implement the *volume calculation* step. We use registers to store the pixel vectors selected as endmembers until the moment, their positions in the image and their volume, and also the current pixel vector data, his position, his 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 Gauss elimination method:

First, for $j = 2, ..., n$ we take a multiple a_{j1}/a_{11} of the first row and subtract it to the j-th row, to make $a_{j1} = 0$. Thus, we have knocked out all elements of matrix $\mathbf{A}$ below the 'pivot' element a_{11} in the first column. Now, for $j = 3, ..., n$, we take a multiple a_{j2}/a_{22} of the second row and subtract it to the j-th row. When we have finished this, all sub-diagonal elements in the second column are zero, and we are ready to process the third column. Applying this process to columns $i = 1, ..., n-1$ completes the matrix triangulation process and matrix $\mathbf{A}$ has been reduced to upper triangular form. These operations are carried out by the data path (see Figure 3).

Obviously, if one of the diagonal pivots a_{ii} is zero, we cannot use a_{ii} to knock out the elements below it; we cannot change a_{ji} by subtracting any multiple of $a_{ii} = 0$ to it. We must switch row i with another row k below it, which contains a nonzero element a_{ki} in the i-th column. Now the new pivot a_{ii} is not zero, and we can continue the matrix triangulation process. If $a_{ki} = 0$ for $k = i, ..., n$, then it will not be satisfactory to switch row i with any of rows below it, as all the potential pivots are zero and therefore $\det \mathbf{A} = 0$. This behaviour has been implemented using a modified circular queue with a small control unit (see Figure 7).

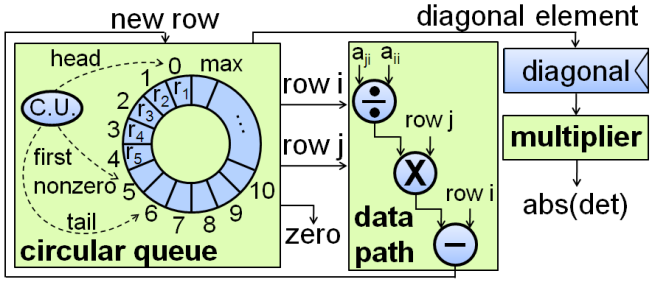


Figure 7. Hardware architecture of the abs(det) module.

Table 2. Summary of Resource Utilization for the FPGA-Based Implementation of the N-FINDR Algorithm For Different Numbers of Endmembers on a Virtex-4 XC4VFX60

Component	Number of endmembers	Number of DSP48Es	Number of slices	Maximum frequency
N-FINDR Module	9	92	6231 (24%)	43.1 MHz
	16	128	11700 (46%)	42.9 MHz
	18	128	14056 (55%)	42.8 MHz
	19	128	17577 (69%)	42.6 MHz
	21	128	24622 (97%)	42.3 MHz
RS232 Transmitter	-	0	71 (0.28%)	208 MHz
DMA Controller	-	0	367 (1.4%)	102 MHz

Finally, the multiplier calculates the multiplication of the main diagonal elements of the triangular matrix and obtains the absolute value.

With this implementation we can extract up to 21 endmembers (the typical number of endmembers per scene lies below this range) with almost total use of available resources of a small FPGA (all embedded DSP48Es multipliers and 97% of the FPGA slices). Table 2 shows the resources used for our hardware implementation of the proposed N-FINDR algorithm design for different numbers of endmembers to be extracted, conducted on the Virtex-4 XC4VFX60 FPGA of the ML410 board.

5. EXPERIMENTAL RESULTS

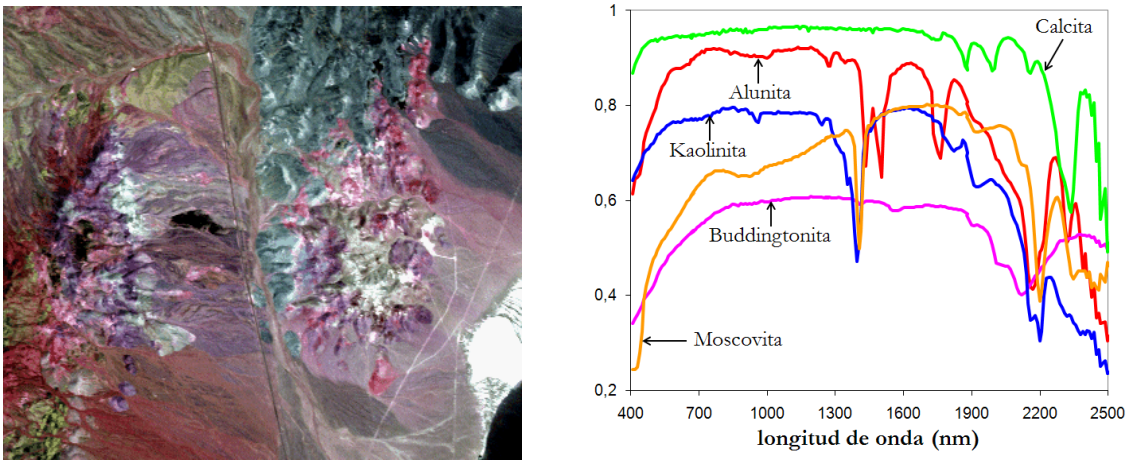
In this section we illustrate the endmember accuracy and performance of the proposed FPGA implementations. The section is organized as follows. In subsection 5.1 we describe the FPGA board used in our experiments. Subsection 5.2 describes the hyperspectral data sets that will be used for demonstration purposes. Subsection 5.3 evaluates the endmember extraction accuracy and execution times of the considered implementations and performs an inter-comparison.

5.1 FPGA Architecture

The hardware architecture described in section 4 has been implemented using the VHDL language. Further, we have used the Xilinx ISE environmentand the Embedded Development Kit (EDK) environmentto specify the complete system. The full system has been implemented on a ML410 board , a low-cost reconfigurable board with a single Virtex-4 XC4VFX60 FPGA component, a DDR2 SDRAM DIMM slot which holds up to 2GBytes, a RS232 port, and some additional components not used in our implementation. We use a Xilinx Virtex-4 XC4VFX60 FPGA because is based on the same architecture as other FPGAs[16] that have been certified by several international agencies for space operation. This FPGA is very close to the space-grade Virtex-4QV XQR4VFX60 FPGA so we could easily implement our design on it.

5.2 Hyperspectral Image Data Sets

Several different hyperspectral data sets have been used in our experiments:



(a) (b)

Figure 8. (a) False color composition of the AVIRIS hyperspectral over the Cuprite mining district in Nevada. (b) U.S. Geological Survey mineral spectral signatures used for validation purposes.

- The first one is the well-known AVIRIS Cuprite scene [see Fig. 8(a)], collected in the summer of 1997 and available online in reflectance units after atmospheric correction. The portion used in experiments corresponds to a 350×350-pixel subset of the sector labeled as f970619t01p02_r02_sc03.a.rfl in the online data which comprises 224 spectral bands in the range from 400 to 2500 nanometers, and a total size of around 50 Megabytes. Bands 1-3, 105-115, and 150-170 were removed prior to the analysis due to water absorption and low SNR in those bands. The site is well understood mineralogically, and has several exposed minerals of interest including *alunite*, *buddingtonite*, *calcite*, *kaolinite* and *muscovite*. Reference ground signatures of the above minerals [see Fig. 8(b)], available in the form of a U.S. Geological Survey library (USGS)will be used to estimate endmember extraction accuracy in this work.

- Second, we have used a set of two AVIRIS images taken over the Jasper Ridge Biological Preserve in California. The datasets are available in both radiance (uncorrected) and reflectance (atmospherically corrected) units. Each of the data sets, acquired on April 1998, consist of 512×614 pixels and 224 spectral bands (for a total size of around 140 Megabytes each). Water absorption and low SNR bands were removed prior to the analysis. In a previous study of surface materials over this area, image endmembers were derived from the scenes above based on extensive ground knowledge.[17] Fig. 9 plots spectral signatures in radiance and reflectance units associated to the main constituent materials at Jasper Ridge. These signatures, corresponding to materials such as *soil*, *evergreen forest*, *dry grass*, *chaparral vegetation*, and *shade*, were obtained from the image scene by using a hybrid method combining visual inspection and prior information about the scene. The location of these materials is also identified in Fig. 9. Ground knowledge was used to identify homogeneous vegetation, shadow and soil areas in the scene. Inside those areas, representative pixels were selected as ground-truth spectra by comparing them to a spectral library of field data, used to represent landscape components at Jasper Ridge. In this process, we ensured that library spectra matched the phenology at the time of the image, and that there was little mis-calibration between field spectra and image spectra.

5.3 Inter-comparison of the hardware PPI and N-FINDR implementations

In this subsection we will make a comparison between the implementations proposed in section 4 of the PPI and N-FINDR algorithms for endmember extraction. A dimensionality reduction transformation is necessary for subsequent execution of the N-FINDR algorithm, however, this step is not necessary for the implementation of PPI algorithm. Dimensional reduction allows us to reduce processing time but often discards relevant information in the spectral domain.

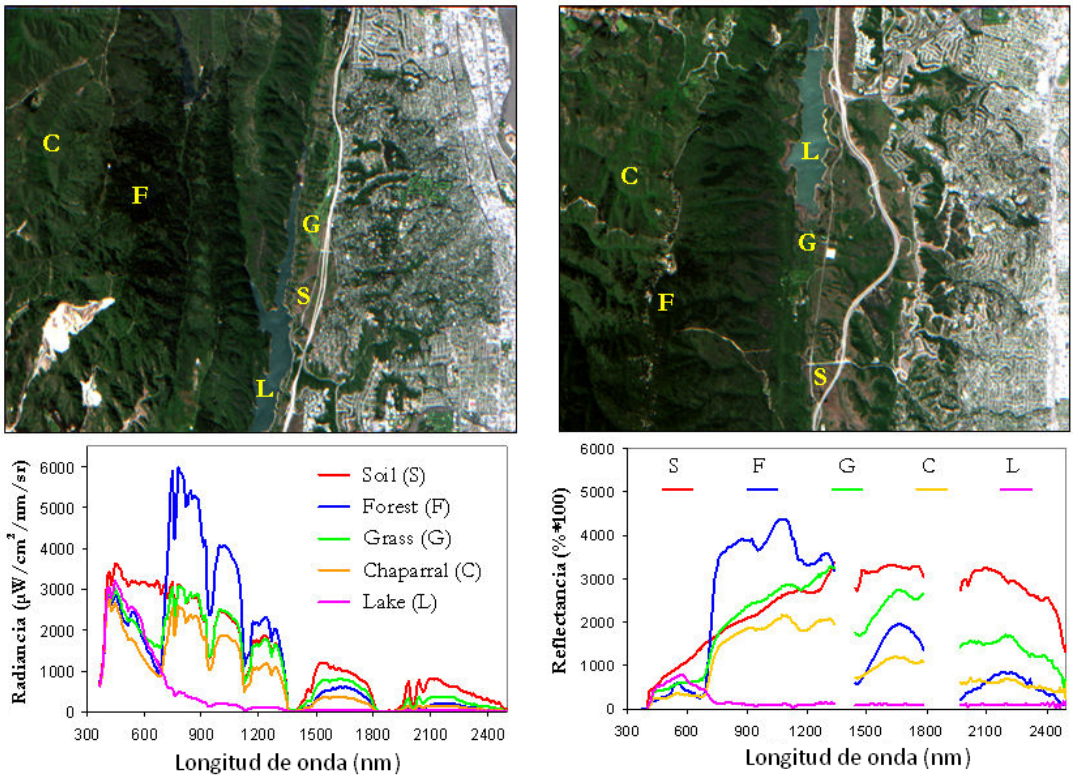


Figure 9. AVIRIS hyperspectral images collected over Jasper Ridge Biological Preserve in radiance (left) and reflectance (right) units, along with the spectral signatures and spatial location of representative endmembers in the two considered scenes.

Table 3. Spectral Angle-Based Similarity Scores between the Endmembers Extracted by our PPI and N-FINDR implementations and the Selected USGS Reference Signatures

	PPI		**N-FINDR**	
	Radians	Degrees	Radians	Degrees
Alunite	0.084	4.812	0.084	4.812
Buddingtonite	0.073	4.182	0.089	5.099
Calcite	0.092	5.271	0.105	6.016
Kaolinite	0.136	7.792	0.138	7.906
Muscovite	0.092	5.271	0.108	6.187

To make a comparison as fair as possible, we performed the dimensional reduction of the images using the Principal Component Analysis (PCA) of the ENVI 4.0 software to make the endmmebers extraction with them. Tables 3 and 4 show the SA for the most similar endmembers extracted by PPI and N-FINDR algorithms over AVIRIS Cuprite scene reduced to 15 superbands and in the AVIRIS Jasper Ridge scene reduced to 18 superbands in both reflectance and radiance units. Moreover, Table 5 shows the computation time of the PPI and N-FINDR algorithms for the different hyperspectral scenes once they have been dimensionally reduced and after the evaluation of 10^4 skewers. Since the processing times of the AVIRIS Jasper Ridge data in radiance and reflectance units are identical, only one is shown in the table.

Firstly, we will make a quantitative comparison between the endmembers extracted by the proposed implementations of the PPI and N-FINDR algorithms. Observing Tables 3 and 4 we realize that, although the values of SA in both algorithms show a high spectral correspondence with the availables reference spectral signatures in every scene, the values obtained by the PPI algorithm are smaller than the values obtained by the N-FINDR algorithm, which means greater spectral similarity. Regarding the response times of both algorithms in processing dimensionally reduced images observing Table 5, it is clear that the PPI algorithm is faster than the N-FINDR algorithm with a speedup slightly higher to 12.

Table 4. Spectral Angle Similarity Scores between the Endmembers Extracted by PPI and N-FINDR from the AVIRIS Jasper Ridge Scenes (in Radiance and Reflectance Units) and the Available Pure Spectral Signatures in Both Scenes.

		Radiance data				
		Soil	Forest	Grass	Chaparral	Lake
PPI	Radians	0.065	0.061	0.045	0.042	0.032
	Degrees	3.724	3.495	2.578	2.406	1.833
N-FINDR	Radians	0.077	0.065	0.044	0.050	0.032
	Degrees	4.411	3.724	2.521	2.864	1.833
		Reflectance data				
		Soil	Forest	Grass	Chaparral	Lake
PPI	Radians	0.030	0.026	0.024	0.031	0.019
	Degrees	1.718	1.489	1.375	1.776	1.088
N-FINDR	Radians	0.028	0.025	0.022	0.020	0.019
	Degrees	1.604	1.432	1.260	1.145	1.088

Table 5. Processing Times Measured for the Hardware PPI and N-FINDR Implementations.

	AVIRIS Cuprite	**AVIRIS Jasper Ridge**
Number of endmembers	16	19
Total size (Megabytes)	50	140
Execution time for PPI Implementation (seconds)	1.35	3.48
Execution time for N-FINDR Implementation (seconds)	13.46	49.35
Speedup	12.48	12.44

Although both algorithms obtain a set of endmembers signatures similar to reference spectral signatures, we must consider that the endmembers extracted by PPI algorithm has greater similarity with such spectral signatures, but most important, it makes the endmember extraction more quickly. Moreover, the N-FINDR algorithm has a set of limitations: One is determining the number of endmembers necessary to be extracted by the N-FINDR algorithm. Another is its computational complexity as a result of an exhaustive search. The third and probably the most critical issue is the requirement of the dimensional reduction of the scene, a process that is often very complex and therefore significantly increases the computation time. A fourth is the use of random initial endmembers resulting in the selection of a final set of endmembers that can be inconsistent and that the results are not reproducible. On the other hand, the N-FINDR algorithm is most popular than the PPI algorithm and PPI needs a post-processing step to obtain the final endmembers from the purity values associated to each pixel, while N-FINDR gives the final endmembers without an aditional step. For all these reasons, we recommend to use the PPI algorithm to implement an unmixing chain in FPGA, since the problem of determining the number of endmembers to be extracted is reduced to determining a threshold value and is not necessary a dimensional reduction of the scene (we do not lose spectral information). In addition, for larger FPGAs PPI algorithm is more easily parallelizable since the basic processing unit uses less hardware resources.

6. CONCLUSIONS AND FUTURE RESEARCH LINES

The number of remote sensing applications requiring fast response of algorithm analysis has been growing exponentially in recent years. Current sensor design practices can greatly benefit from the inclusion of radiation-hardened FPGAs, which can be easily mounted or embedded in the sensor due to its compact size and low-weight, which does not compromise mission payload. In this paper, we present an inter-comparison of the the experimental results of our FPGA implementation of the PPI and N-FINDR algorithms, the most well-known approaches for hyperspectral data analysis in the remote sensing community. Our experimental results, conducted on a Virtex-4 XC4VFX60, demonstrate that our architecture can extract endmembers with highly satisfactory spectral purity. To implement an unmixing chain in FPGA, we recommend to use the PPI algorithm since is not necessary a dimensional reduction and provides better performance. Further, our proposed hardware version of the algorithms can significantly outperform (in terms of computation time) an equivalent software version.

As future work, we are investigating FPGA implementations of techniques for estimating the number of endmembers in the scene, such as the virtual dimensionality concept in,[18] as well as techniques for estimating

endmember abundances in order to provide a full spectral unmixing chain.

ACKNOWLEDGMENTS

This work has been supported by the European Community's Marie Curie Research Training Networks Programme under reference MRTN-CT-2006-035927, Hyperspectral Imaging Network (HYPER-I-NET). This work has also been supported by the Spanish Ministry of Science and Innovation (HYPERCOMP/EODIX project, reference AYA2008-05965-C04-02), AYA2009-13300-C03-02 y TIN2009-09806. The authors gratefully thank Dr. Robert O. Green at NASA/JPL for providing the AVIRIS data sets used in our experimental assessment.

REFERENCES

[1] Goetz, A. F. H., Vane, G., Solomon, J. E., and Rock, B. N., "Imaging spectrometry for Earth remote sensing," *Science* **228**, 1147–1153 (1985).

[2] Green, R. O. et al., "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," *Remote Sensing of Environment* **65**(3), 227–248 (1998).

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

[4] El-Ghazawi, T. A., El-Araby, E., Huang, M., Gaj, K., Kindratenko, V. V., and Buel, D. A., "The promise of high-performance reconfigurable computing," *IEEE Computer* **41**, 69–76 (2008).

[5] Plaza, A., Martinez, P., Perez, R., and Plaza, J., "A quantitative and comparative analysis of endmember extraction algorithms from hyperspectral data," *IEEE Transactions on Geoscience and Remote Sensing* **42**(3), 650–663 (2004).

[6] Heinz, D. and Chang, C.-I., "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing* **39**, 529–545 (2000).

[7] Plaza, A. and Chang, C.-I., [*High Performance Computing in Remote Sensing*], Taylor & Francis: Boca Raton, FL (2007).

[8] Keshava, N. and Mustard, J. F., "Spectral unmixing," *IEEE Signal Processing Magazine* **19**(1), 44–57 (2002).

[9] Chang, C.-I., [*Hyperspectral Data Exploitation: Theory and Applications*], John Wiley & Sons: New York (2007).

[10] Winter, M. E., "N-FINDR: an algorithm for fast autonomous spectral end-member determination in hyperspectral data," *Proc. SPIE Image Spectrometry V* **3753**, 266–277 (2003).

[11] Green, A. A., Berman, M., Switzer, P., and Craig, M. D., "A transformation for ordering multispectral data in terms of image quality with implications for noise removal," *IEEE Transactions on Geoscience and Remote Sensing* **26**, 65–74 (1988).

[12] Schowengerdt, R. A., [*Remote Sensing: Models and Methods for Image Processing, 2nd ed.*], Academic Press: New York (1997).

[13] Lavernier, D., Fabiani, E., Derrien, S., and Wagner, C., "Systolic array for computing the pixel purity index algorithm on hyperspectral images," *Proceedings of SPIE* **4480**, 130–138 (1999).

[14] Lavernier, D., Theiler, J., Szymanski, J., Gokhale, M., and Frigo, J., "FPGA implementation of the pixel purity index algorithm," *Proceedings of SPIE* **4693**, 30–41 (2002).

[15] Plaza, A. and Chang, C.-I., "Clusters versus FPGA for parallel processing of hyperspectral imagery," *International Journal of High Performance Computing Applications* **22**(4), 366–385 (2008).

[16] " Xilinx. Available online: http://www.xilinx.com/publications/prod_mktg/ virtex5qv-product-table.pdf,"

[17] Garcia, M. and Ustin, S. L., "Detection of interannual vegetation responses to climatic variability using AVIRIS data in a coastal savanna in California," *IEEE Transactions on Geoscience and Remote Sensing* **39**, 1480–1490 (2001).

[18] Du, Q. and Chang, C.-I., "Estimation of number of spectrally distinct signal sources in hyperspectral imagery," *IEEE Transactions on Geoscience and Remote Sensing* **42**(3), 608–619 (2004).