A Scalable and Dynamically Reconfigurable FPGA-Based Embedded System for Real-Time Hyperspectral Unmixing

Teresa G. Cervero, Julián Caba, Sebastián López, Member, IEEE, Julio Daniel Dondo, Roberto Sarmiento, Fernando Rincón, Member, IEEE, and Juan Carlos López, Member, IEEE

Abstract—Earth observation hyperspectral imaging instruments capture and collect hundreds of different wavelength data corresponding to the same surface. As a result, tons of information must be stored, processed, and transmitted to ground by means of a combination of time-consuming processes. However, one of the requirements of paramount importance when dealing with applications that demand swift responses is the ability to achieve real-time. In this sense, the authors present a flexible and adaptable Field-Programmable Gate Array (FPGA)-based solution for extracting the endmembers of a hyperspectral image according to the Modified Vertex Component Analysis (MVCA) algorithm. The proposed approach is capable of adapting its parallelization execution by scaling the execution in hardware. Thus, the solution uses the dynamic and partial reconfiguration property of FPGAs in order to exploit and vary the level of parallelism at run-time. In order to validate the convenience of using this kind of solutions, the performance of our proposal has been assessed with a set of synthetic images as well as with the well-known Cuprite hyperspectral image. The achieved results demonstrate that the proposed system might be dynamically scaled without significantly affecting total execution times, being able to extract the endmembers of the Cuprite dataset in real-time.

Index Terms—Dynamic reconfiguration, Field-Programmable Gate Array (FPGA), hyperspectral images, linear unmixing, Modified Vertex Component Analysis (MVCA).

I. INTRODUCTION

HYPERSONTAL imaging has become a popular technique in many fields of science and engineering. These images are sampled in a high number of wavelengths, resulting in data cubes containing hundreds or even thousands of nearly contiguous spectral bands. By following this strategy, hyperspectral images provide much more information about the captured scene than traditional solutions based on panchromatic or multispectral approaches. This fact has favored the exploitation of hyperspectral data for many remote sensing applications such as environmental monitoring, military surveillance, or rare mineral detection, among many others.

The concept of hyperspectral imaging was originated at NASA’s Jet Propulsion Laboratory (JPL) in California, which developed instruments such as the airborne visible infrared imaging spectrometer (AVIRIS). These instruments are able to record the visible and near-infrared spectra of the reflected light of an area from 2 to 12 km wide and several kilometers long using 224 spectral bands [1]. Due to the low spatial resolution provided by these devices, several spectrally pure distinct materials (also called endmembers) can be found as a part of the same pixel. Thus, each pixel can be viewed as a mixture of the endmembers signatures [2]. A classical methodology for analyzing this kind of images is based on linear unmixing models, which considers that these mixed pixels are a linear combination of the endmember signatures present in the scene weighted by the correspondent abundance fractions (i.e., the percentage of each endmember).

Within the wide spectra of linear unmixing algorithms, the vertex component analysis (VCA) [3] is one of the most successful and popular since it reaches better results than others. This algorithm is very intensive in computation, and this is the reason because it is a well-suited candidate to be accelerated on hardware. In addition, the Modified Vertex Component Analysis (MVCA) algorithm [4] behaves very similar than the VCA algorithm, but simplifying some of the VCA operations. In the end, the MVCA algorithm has demonstrated to outperform the VCA algorithm in terms of endmember extraction accuracy versus computational complexity [4]. Unfortunately, for applications under real-time or near real-time constraints, the MVCA still needs to be further accelerated onto a high-performance computing platform. Parallel computing techniques have been widely used to accelerate hyperspectral imaging algorithms during the last years [5]. More specifically, implementations on specialized hardware devices, such as Field-Programmable Gate Arrays (FPGAs) [6] and Graphics Processing Units (GPUs) [7] have been recently proposed. Over the last few years, reconfigurable hardware solutions, such as FPGAs, have been consolidated as the standard choice for space-based onboard data processing platforms [4]. Their success lies on their smaller size and weight compared with traditional cluster-based systems, as well as to their...
lower power dissipation figures when they are compared with GPUs [8]. Furthermore, the increasing number of FPGAs with tolerance to ionizing radiation in space turns these devices into robust hardware solutions.

FPGAs are formed by blocks of logic circuits and a massive set of communication channels used to connect these blocks to form components with a specific behavior. Then, an FPGA is a programmable device in which its internal structure can be configured (programmed) to perform specific logic tasks. Due to this issue and their flexibility and performance characteristics, FPGAs allows different granularities of parallelism that can be used to implement different types of algorithms exploiting parallel processing. Therefore, FPGAs have promoted their use as a suitable candidate for running high-performance computing applications due to their low power consumption and reduced time-to-market, among other features. In fact, the inherent parallelism and the run-time reconfigurability feature offered by these devices open up a wide range of possibilities to deploy a new generation of scalable solutions capable of coping with the stringent demands of real-time hyperspectral imaging applications. In this sense, the exploitation of the scalability at run-time allows the system to be adapted to the input image characteristics, or even to other external demands, by modifying the number of resources involved in the data processing. These dynamic variations of the available number of resources have several consequences. The first one is that it is necessary to implement a flexible parallelization strategy. Second, the execution time might be adapted according to the level of scalability. The last but not the least comes from the possibility of saving area, resources, power consumption, increasing the reusability of the available resources, and also increasing the robustness of the system against faults. In particular, this paper presents a dynamically reconfigurable implementation of the MVCA endmember extraction algorithm on an FPGA platform, which might vary its level of parallelism by modifying the number of processing elements involved in the execution. As a result, a run-time scalable FPGA-based MVCA system is developed.

The rest of this paper is organized as follows. Section II introduces the basis of the VCA and the MVCA endmember extraction algorithms. Then, Section III describes the main characteristics of three MVCA hardware approaches designed onto an FPGA, where two of them allow scaling their computing resources. Section IV explains how to successfully control and manage the dynamic and partial reconfiguration process. Section V presents the final system as part of a hardware platform and evaluates its performance. Finally, Section VI outlines the conclusion of this paper.

II. VCA AND MVCA ENDMEMBER EXTRACTION ALGORITHMS

Linear unmixing models consider that each pixel \( y \in \mathbb{R}^L \) (where \( L \) is the number of bands) of a hyperspectral image composed by \( R \) observed pixels, can be obtained as

\[
y = Ea + n
\]  

where \( E \equiv [e_1, e_2, \ldots, e_p] \) is a full-rank \( L \times p \) mixing matrix (\( e_j \) denotes the \( j \)th endmember signature), \( p \) is the number of endmembers present in the image (with \( p < L \)), \( a = [a_1, a_2, \ldots, a_p]^T \) is the abundance vector containing the fractions of each endmember (notation \((\cdot)^T\) stands for transpose vector), and \( n \) is an additive noise vector.

The procedure of linearly unmixing a given hyperspectral image tries to first accurately estimate the number of endmembers (\( p \)), then to extract their spectral signatures (\( E \)), and finally, to obtain their respective abundance fractions (\( a \)). During the last decade, a huge amount of algorithms have been proposed for each of these three stages. Regarding the endmember extraction stage, the most widely used class of algorithms adopts a geometrical framework, which assumes that there is at least one spectral vector on each vertex of the data simplex. Graphically, this means that each pixel is represented in a space formed by \( L \) axis (one axis per spectral band), and the endmembers are the most extreme pixels which enclose to all the rest of pixels of the image. This representation is known as a simplex. In this sense, interested readers can find an excellent and comprehensive review of all these methods in [2].

According to [4], the MVCA algorithm is able to obtain the same level of performance than the VCA but with simpler operations. Due to these changes, the computational time to obtain the endmembers is greatly reduced. In Section II-A, a brief review of the VCA is presented, whereas in Section II-B, the MVCA algorithm is described in detail, highlighting the main differences with the VCA.

A. VCA Algorithm

VCA is based on the algebraic fact that the endmembers are the vertices of a simplex, being the affine transformation of a simplex also a simplex. It uses a positive cone defined by the hyperspectral data to be processed, which projected on a properly chosen hyperplane gives a simplex with vertices corresponding to the endmembers. After projecting the data onto the selected hyperplane, the VCA algorithm projects all image pixels to a random direction, obtaining the first endmember as the pixel with the largest projection. The other endmembers are identified by iteratively projecting the data onto an orthonormal direction (given by a vector named \( f \)) to the subspace spanned by the endmembers already determined. The new endmember is then selected as the pixel corresponding to the extreme projection, and the procedure is repeated until the whole set of \( p \) endmembers is found.

Within the VCA algorithm, we can distinguish at least two computational processes with a high degree of complexity and as a consequence, as potential candidates to be modified in order to accelerate their execution and to increase the efficiency of an ulterior algorithm implementation. The first one is the process by which vector \( f \) is obtained, which requires computing the pseudoinverse of the matrix where the already computed endmembers are stored. The second one is the process by which the hyperspectral dataset is projected onto the direction pointed by vector \( f \). Since the number of pixels in a hyperspectral image is usually large, this amount of flops may be even
bigger than for the computation of vector \( f \), which implies that a huge number of flops have to be carried out per iteration at the VCA algorithm.

### B. MVCA Algorithm

MVCA algorithm outperforms the VCA by applying a low-complexity orthogonalization method and utilizing integer instead of floating-point arithmetic when dealing with hyperspectral data. This is achieved by modifying the processes by which vector \( f \) is calculated and by which the hyperspectral image is projected onto the direction pointed by vector \( f \), as it will be explained as follows.

Two main modifications are introduced in the process for computing the vector \( f \). First, the norm of this vector is not forced to be equal to the unity, which means that the operations performed in the VCA algorithm in order to normalize vector \( f \) are skipped in the MVCA algorithm. The second and the most important modification in terms of computational cost savings and ease of implementation is based on changing the mechanism adopted in the VCA algorithm for the calculation of a vector orthogonal to the subspace spanned by the endmembers that have been already determined. In particular, \( f \) is computed by first obtaining an orthogonal set of \( i \) vectors

\[
U = \{u_1, u_2, \ldots, u_i\}
\]

from the set \( E = \{e_1, e_2, \ldots, e_i\} \), defined by the endmembers that have been already computed. This is achieved by applying the Gram–Schmidt orthogonalization algorithm, which guarantees that the set \( U \) spans the same \( i \)-dimensional subspace as \( E \)

\[
u_k = e_k - \sum_{j=1}^{j=k-1} \text{proj}(e_k, u_j), \quad \{k = 1 \ldots i, \text{ and } u_1 = e_1\}
\]

(2)

where the proj stands for the projection operator, defined as

\[
\text{proj}(e_k, u_j) = u_j \cdot \frac{\langle e_k, u_j \rangle}{\langle u_j, u_j \rangle}
\]

where \( \langle x, y \rangle = \sum_{z=1}^{z=P} x_z \cdot y_z \).

(3)

As far as \( U \) spans the same \( i \)-dimensional subspace of \( E \), an additional vector \( u_{i+1} \) is also orthogonal to all the vectors included in \( E \) and \( U \), avoiding the computation of the pseudoinverse of matrix \( E \). Vector \( u_{i+1} \) is computed by following the procedure stated by

\[
f = w - \sum_{l=1}^{l=i} \text{proj}(w, u_l).
\]

(4)

Once \( f \) has been computed, the hyperspectral image \( Y \) must be projected onto the direction indicated by this vector. In order to further reduce the computational complexity of the endmember extraction process, this projection is performed in the MVCA algorithm using integer rather than floating point arithmetic. This criterion is based on the idea that this modification should not alter the position of the projection extreme (although the value of the projection itself will definitively change).

### III. Scalable Architectures

Traditionally, linear unmixing algorithms have been simulated or emulated using software tools. However, these kinds of solutions are not enough for covering the real-time exigencies imposed by some remote sensing applications. In particular, the endmember extraction stage is one of the most data intensive stages, as part of the linear unmixing processing chain. Therefore, it can become a bottleneck for coping with real-time constraints. This section explores how to accelerate the execution of the MVCA algorithm [9] in hardware using different and flexible parallelization strategies on FPGAs. In order to appreciate the main differences between a traditional static and parallel solution compared to a scalable one, the following Sections III-A–III-C present three approaches on how to implement the MVCA algorithm in hardware onto an FPGA. The former is a common Simple Instruction Multiple Data (SIMD) parallel approach (StaticArch_MVCA) that can be considered static once its parameters are fixed. On the contrary, the next two ones use the scalability as a mechanism for adapting the data level parallelism (DLP) without constraining the solutions, though from different perspectives. One of them parallelizes the number of spectral components that might be processed simultaneously (SpectScalableArch_MVCA), whereas the other one might compute several pixels concurrently (SpatialScalableArch_MVCA).

#### A. Static MVCA Architecture: StaticArch_MVCA

The goal of this hardware design is to extract the endmembers present into a hyperspectral image. In order to do this, the design follows an iterative process, where the number of iterations directly depends on the image’s characteristics. More specifically, the number of iterations is determined by the number of pure pixels (also known as endmembers) that exist in the image. A first modular and hardware approach that behaves according to the MVCA algorithm is depicted in Fig. 1. The structure is based on several modules dedicated to specific tasks (computation, data loading and/or storage, and data format conversion). The most important ones are focused on the computation, which are the \( U_{\text{GENERATOR}} \), the \( F_{\text{GENERATOR}} \), and the IMAGE PROJECTION modules. There is another module (INPUT MEMORY) in which the hyperspectral image is stored as a matrix. Here, the two spatial dimensions are represented in columns, and the spectral components of a pixel correspond to the rows. At the end of an iteration, an endmember is addressed through an integer variable (index) as the output of the IMAGE PROJECTION module. This last module calculates the projection of all the pixels that compose the hyperspectral image onto the direction pointed by the vector \( f \). This vector is computed in the \( F_{\text{GENERATOR}} \) module, as an orthogonal vector to the set of vectors computed in the \( U_{\text{GENERATOR}} \) module. These vectors are obtained from the endmembers that have been already calculated using a Gram–Schmidt orthogonalization. Comparing these three modules, the IMAGE PROJECTION is the one in charge of processing the whole hyperspectral image, whereas the rest of them only process certain values. Furthermore, its computation is based on a matrix multiplication.
between the vector \( f \) and the hyperspectral image, according to the following expression:

\[
f^T \times Y_{int} = (f_1 f_2 \ldots f_p) \times \begin{bmatrix} y_{11} y_{12} \ldots y_{1R} \\ y_{21} y_{22} \ldots y_{2R} \\ \vdots \\ y_{p1} y_{p2} \ldots y_{pR} \end{bmatrix} = C_k
\]

\[
C_k = \sum_{i=1}^{p} f_i \times y_{ij} \quad \forall j = 1, 2, \ldots, R
\]

\( \text{index} = MemPosition(\max \text{proj}(C_k)). \) (7)

The regularity of these operations and the low data dependences make easy to exploit DLP, providing a high flexibility to perform several operations simultaneously. In this sense, this approach might be considered static once the number of spectral bands of a pixel is fixed. This is because the number of hardware resources will remain the same after the configuration of the FPGA. An immediate consequence of this fact is that, independently of the system or the environmental requirements, the logic resources cannot be modified while the system is running. By attending to these premises, the IMAGE PROJECTION module is based on multipliers and the subsequent adder-tree as shown in Fig. 2. In order to accelerate as much as possible the calculation of every pixel projection \( (C_k) \), the image projection dedicates all its resources to compute a whole projection per clock cycle. In this way, the number of multipliers has to be the same than the number of spectral bands of a pixel \( (p) \). Every multiplier is responsible of one spectral component, and then, the pixel projection is completed by the adder-tree.
B. Scalable MVCA Architectures

Despite the fact that the StaticArch_MVCA architecture takes advantage of the DLP paradigm, the fact is that it is not flexible enough for easily adapting the performance to the variations of the rest of the system or even the environment. This is because of using a fixed number of dedicated resources, once it has been configured. Therefore, in case that the system requirements are relaxed, part of its resources will remain idle and without having the alternative to be used by any other task running on hardware. This is the reason why this subsection is focused on exploring how to introduce a higher flexibility to the IMAGE PROJECTION computation. Before going deeper on the solution, it is relevant to mention that the scalability efforts are only applied in the IMAGE PROJECTION module, since this is the most critical task and it can become a bottleneck on the system’s computation due to the large amount of memory accesses required to compute the pixels’ projection. Consequently, the rest of the functional modules belonging to the MVCA (INPUT MEMORY, _U_GENERATOR_, and _F_GENERATOR_) are the same for the two approaches presented below.

The IMAGE PROJECTION module’s behavior and its data dependences allow to parallelize the computation by using two different strategies, both of them explained along the following sections, and respectively named as spectral and the spatial scalability. The former processes one pixel projection at a time and its scalability impacts on the number of spectral components that are simultaneously computed. The latter scales the number of pixels’ projections that are concurrently computed. These policies introduce a higher flexibility to the solution by exploiting run-time scalability in hardware. The scalability level impacts on the number of PEs involved into the computation, and this number can vary between one and \( p \) (corresponding to the number of spectral bands of the image’s pixel after the dimensionality reduction).

1) Spectral Scalability: SpectScalableArch_MVCA: This architecture takes advantage of a spectral parallelization strategy which allows introducing a higher flexibility into the computation of the IMAGE PROJECTION module.

The overall structure of this approach is similar to the StaticArch_MVCA one, but including some modifications. The static array of multipliers has been substituted by a set of PEs, and the adder-tree has reduced its number of hierarchy levels, as shown in Fig. 3. Furthermore, the main difference of this scalable solution, when compared to the static one, is the fact that the number of PEs is not fixed. On the contrary, the amount of this kind of elements is dynamic and can be scaled according to the system’s requirements. Thus, the Scalable PEs (Fig. 3) module is formed by replicas of the same element (PE). Due to the fact that the functionality and the behavior of all the PEs are the same, the workload is distributed between them regularly. As a result, the available PEs start processing at the same time, but manipulating different spectral components of the same pixel. In the end, the adder-tree completes the pixel projection by adding the outputs of all the PEs. Table I shows a step-by-step execution when the number of PEs (\( nPEs \)) is two, and the number of spectral components (\( p \)) is odd. The relationship between these two values (\( nPEs \) and \( p \)) determines the number of times (\( Iterations_{PE} \)) that a PE has to process different components of every pixel.

\[
Iterations_{PE} = \begin{cases} 
\frac{nPEs}{p}, & \text{when } (nPEs \mod p) = 0 \\
\frac{nPEs}{p} + 1, & \text{when } (nPEs \mod p) \neq 0.
\end{cases}
\] (8)

Moreover, in the last iteration, some PEs might remain idle, since they have to wait until all the PEs complete the pixel projection computation in order to keep the system synchronized. The number of idle PEs into the last iteration corresponds to

\[
Idle_{PEs} = nPEs - (nPEs \mod p).
\] (9)

When a pixel projection has been performed, the process is repeated again, and it continues until the whole pixels’ projections have been calculated. As far as the idea of including more PEs than the number of spectral components of a pixel is useless, (due to the behavioral description of this approach) the level of scalability is limited between one and \( p \). Otherwise, the extra number of PEs will remain idle during the whole execution. As it can be seen along this section, in this context, the
TABLE I

<table>
<thead>
<tr>
<th>Module</th>
<th>Iterations</th>
<th>PE₀</th>
<th>PE₁</th>
<th>Accumulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>PES</td>
<td>1</td>
<td>(f₁ \times y₁₁ = a₀)</td>
<td>(0 + a₀)</td>
<td>(f₁ \times y₁₁ = b₀)</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>(f₂ \times y₂₁ = a₁)</td>
<td>(a₀ + a₁)</td>
<td>(f₂ \times y₂₁ = b₁)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>(f₃ \times y₃₁ = a₂)</td>
<td>(a₀ + a₁ + a₂)</td>
<td>(f₃ \times y₃₁ = b₂)</td>
</tr>
<tr>
<td></td>
<td>(n)</td>
<td>(fₙ \times yₙ₁ = aₙ)</td>
<td>(\sum_{i=0}^{n-1} a_i)</td>
<td>(\text{idle})</td>
</tr>
<tr>
<td>Adder-tree</td>
<td>(n + 1)</td>
<td>(Cₖ = \sum_{i=0}^{n-1} a_i + \sum_{i=0}^{n-1} b_i)</td>
<td>(Cₖ)</td>
<td></td>
</tr>
<tr>
<td>Index</td>
<td>(n + 2)</td>
<td>(\text{Process a new pixel by repeating the process from 1})</td>
<td>(\text{index}<em>{\text{max}} = \max(\text{index}</em>{\text{max}}, Cₖ))</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 4. Detailed IMAGE PROJECTION module of the SpatialScalableArch_MVCA design.

scalloplcity refers to the property that permits adjusting how fast a result is obtained without influencing on the result itself. Particularly, in this paper, the scalability is exploited by replicating the same kind of element (PE) and then making possible that all of them collaborate together in the computation of a specific task (IMAGE PROJECTION).

This SpectScalableArch_MVCA architecture can be characterized in terms of its benefits and drawbacks when compared to the StaticArch_MVCA architecture. The most remarkable strengths are related to the flexibility, hardware reusability, and adaptability, since the performance might vary according to the number of PEs. However, not everything is favorable when this type of scalability is exploited. For example, due to the parallelization strategy, this scalable architecture has some limitations concerning the performance. In this sense, it is desirable than the performance, referred to the number of clock cycles required for calculating the index value, improves when the number of PEs also increases. Unfortunately, in this case, it is not always true. Due to the fact that the computation of a pixel projection needs the same number of clock cycles when \(\frac{n}{2} < nPE < p\), though the number of hardware resources changes. Another inconvenient of this approach occurs when \(nPE = 1\). Here, the whole pixel projection is completely performed by the PE itself, and consequently the adder-tree would not be necessary. Nevertheless, the adder-tree is always present as part of the system, as it is static and it never varies.

Concerning the synchronization, the most effective way of implementing this behavioral strategy is to guarantee that all the PEs start processing simultaneously. In this way, it is possible to get their results at the same time, and then completing the pixel projection calculation by adding their values.

2) Spatial Scalability: SpatialScalableArch_MVCA: Another way to face the IMAGE PROJECTION module’s computation is using a spatial parallelization strategy, in which the processing elements are dedicated to one pixel projection each, instead of being concentrated in computing the same projection. Therefore, a PE is in charge of processing one by one all the spectral components of a whole pixel until completing its projection. In the case that all PEs start processing at one time, then more than one pixel projection would be produced simultaneously. Although it is possible to control this scenario, it implies including a comparator-tree in order to manage and calculate the highest pixel projection. As an alternative, this SpatialScalableArch_MVCA approach delays one clock cycle the execution of consecutive processing elements, according to the scheme shown in Fig. 4. Under these specifications, and considering only two PEs, a step-by-step execution will follow the pattern presented in Table II. It is possible to observe how a PE has to iterate \(p\) times, which is the same number than the spectral components of a pixel. Besides, when a PE completes a pixel projection, automatically it starts with a new pixel without waiting for any other PE.

Regarding the scalability level, in this approach, the number of PEs can be comprised between one and \(p\). In spite of this fact, it would be possible to increase the scalability to higher numbers, but at the cost of introducing a comparator-tree in the system, as it was previously mentioned, since more than one result would be generated.
### TABLE II
**Step-by-Step Execution of the SpatialScalableArch\_MVCA With Two PEs**

<table>
<thead>
<tr>
<th>PES</th>
<th>Module</th>
<th>Iterations</th>
<th>$f_1 \times y_{n-1} = a_n</th>
<th>0 + a_n</th>
<th>f_1 \times y_{n-1} = b_n</th>
<th>0 + b_n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>$a_1$</td>
<td>$a_1 + a_2$</td>
<td>$b_1$</td>
<td>$b_1 + b_2$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>$a_n$</td>
<td>$a_n + a_{n+1}$</td>
<td>$b_n$</td>
<td>$b_n + b_{n+1}$</td>
</tr>
<tr>
<td>$n$</td>
<td></td>
<td></td>
<td>$x_0 = \sum_{i=0}^{n-1} a_i$</td>
<td>$f_p \times y_{p-1} = b_{n-1}$</td>
<td>$x_1 = \sum_{i=0}^{n-2} b_i$</td>
<td></td>
</tr>
<tr>
<td>$n+1$</td>
<td></td>
<td></td>
<td>$\text{index}<em>{\text{max}} = \max (\text{index}</em>{\text{max}} x_0)$</td>
<td>$f_p \times y_{p+1} = b_n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n+2$</td>
<td></td>
<td></td>
<td>$\text{index}<em>{\text{max}} = \max (\text{index}</em>{\text{max}} x_0)$</td>
<td>$x_1 = \sum_{i=0}^{n-1} b_i$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process a new pixel by repeating the process from 1

### TABLE III
**Synthesis Report of the Scalable MVCA Architectures onto a Xilinx V5-LX110T FPGA**

<table>
<thead>
<tr>
<th>HW architecture</th>
<th>Configuration</th>
<th>Synthesis report</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of endmembers</td>
<td>No. of PEs</td>
</tr>
<tr>
<td>StaticArch_MVCA</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>SpectScalableArch_MVCA</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15 863</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>16 222</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>23 355</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>26 094</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>40 022</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>40 764</td>
</tr>
<tr>
<td>SpatialScalableArch_MVCA</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15 331</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>16 023</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>25 175</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>26 930</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>40 095</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>42 685</td>
</tr>
</tbody>
</table>

### C. MVCA Architectural Comparison

This section discloses the performance of the three approaches presented above: the StaticArch\_MVCA, the SpectScalableArch\_MVCA, and the SpatialScalableArch\_MVCA. The performance is expressed in the form of hardware resources occupancy, and the time required to extract a finite set of endmembers from a given hyperspectral image. In the case of the scalable approaches, different configurations have been tested using one or several PEs. Regarding the FPGA, all the tests have been run on a Xilinx FPGA Virtex-5 LX110T, since it is a powerful FPGA with a regular layout, which eases the scalability and the dynamic reconfiguration process.

Table III shows information related to the hardware logic occupancy of the three considered approaches, according to the synthesis report information, when different number of endmembers has to be extracted from different synthetic images and several PEs are available. The occupancy is represented in terms of the number of slice registers (N. OF SLICE REGISTERS), number of slice LUTs (N. OF SLICE LUTS) and number of DSPs (N. OF DSP48E). In addition, the clock frequency has been also included in the last column (FREQUENCY). After analyzing these numbers, in all the cases, the hardware logic resources increase due to two factors: the number of endmembers to extract (directly dependent on the hyperspectral image characteristics), and the number of PEs used for computing the data (directly dependent on the architectural characteristics). Moreover, considering the case in which there is to extract a fixed number of endmembers, the scalable architectures require more resources than the static one. This fact is because of the control and data distribution cost of exploiting the scalability.

Furthermore, in order to analyze the behavior and the reliability of the proposed MVCA scalable implementations, we have selected as a real hyperspectral image the subset of the well-known Cuprite hyperspectral image composed by 250 × 191 pixels and 224 spectral bands shown in Fig. 5. The Cuprite dataset is a well-known hyperspectral image that is considered as a reference within the hyperspectral remote sensing research field. This scene has been captured by NASA’s AVIRIS sensor, which collects data in 224 contiguous spectral bands with a bandwidth of 0.10 μm. Each 20-m square cell in the
scene has a contiguous spectrum over the range from 0.4 to 2.5 μm. Moreover, the selected subset is available online in reflectance units and is well understood from a mineralogical point of view. Due to these reasons, it has become a reference for testing the performance and accuracy for many algorithms within the linear unmixing field, including different endmember extraction algorithms.

Despite the fact that the MVCA algorithm, and also our hardware solution, is able to extract the endmembers of a hyperspectral image with all its spectral bands, the quality and accuracy of the extraction does not vary when the input is dimensionally reduced. Therefore, in order to reduce the memory needs of the final system, and also accelerate the computation, all the images (including the aforementioned Cuprite subset) that have been used as input to the proposed MVCA approach have been preprocessed by limiting its amount of spectral bands to the number of endmembers. Hence, since the number of endmembers present in a real hyperspectral scene such as Cuprite is usually unknown a priori, its value might be calculated prior to the unmixing step. More specifically, we have used the Virtual Dimensionality (VD) algorithm [11] as part of a preprocessing stage. This algorithm determined that there are 14 different pure materials present within the targeted Cuprite subimage. Based on this information, the number of spectral bands has been reduced from 224 bands to 14. Consequently, for these experiments the input image comprises 250 × 191 pixels and 14 spectral bands, which result in a total of 668 500 elements (47 750 pixels × 14 spectral bands).

In order to analyze the impact of the scalability on the performance of the MVCA algorithm, the two scalable architectures (SpectScalableArch_MVCA and SpatialScalableArch_MVCA) are compared in Fig. 6. This chart represents the time required for extracting the 14 endmembers of the Cuprite image, according to the post Place and Route (PaR) report information when different numbers of PEs are used. Under this circumstance, the SpatialScalableArch_MVCA is faster than the other scalable solution, while also improves the computation time when the number of PEs increases in the system.

More in detail, Table IV summarizes the occupancy and the performance of these three architectures once they have been routed onto the selected FPGA. The logic hardware utilization has been considered in terms of number of slice registers (N. OF SLICE REGS.), LUTs (N. OF SLICE LUTS) and DSPs (N. OF DSP48E). The results include the PaR clock frequency (FREQUENCY (MHz)), the number of clock cycles for extracting 14 endmembers (N. OF CLOCK CYCLES) and the final computation time [TIME (ms)] for extracting the 14 endmembers present into this image.

Considering the information reported in Fig. 6 and Table IV, it is possible to identify the SpatialScalableArch_MVCA architecture as the most efficient one in terms of performance and flexibility. In spite of the fact that this design demands more hardware resources than the StaticArch_MVCA in the less favorable scenario (nPE = p = 14) (9.89% more slice registers, and 12.01% more slice LUTs), the SpatialScalableArch_MVCA presents the advantage of being flexible enough for adapting its performance when the environmental conditions are relaxed or the hyperspectral image to be processed reduces or increases its number of endmembers. While the SpatialScalableArch_MVCA scales its number of PEs to these particular cases, the StaticArch_MVCA is completely static once its parameters have been fixed. As a consequence, this fact implies an inefficient design in the sense that some hardware resources may remain unused if the characteristics (number of endmembers) of the image vary.

IV. Dynamic Reconfigurability

Advances on reconfigurable devices have made FPGAs more and more attractive for embedded systems design. Besides the improvement shown in several features; such as the performance, the efficiency, and the number of dedicated resources; the most important FPGAs vendors have included a special characteristic to some of their products: the reconfigurability facility. This one can be organized into two groups: the static, or the dynamic reconfigurability. During the static reconfiguration process the device is not active. In other words, the entire device must be stopped, configured and then brought up after the configuration is completed. This methodology needs additional management to preserve the system status, since before a reconfiguration procedure, the system has to save its context in memory and all its components are stopped, then the reconfigurable process may happen, and after that the previous context is recovered [12]. The latter, the dynamic reconfigurability is more flexible since it permits isolating those regions of the FPGA in which the modifications have to be introduced from those one that remain unaltered.

Considering the scalable scenario proposed in the previous section, in which the performance of the system might be adapted according to the number of PEs that are running on it, this work takes advantage of the dynamic reconfigurability with the goal of introducing those hardware modifications at run-time, without interrupting the execution of the rest of the system. Before going in detail in the physical implementation of the proposed system, the following paragraphs introduce several designing rules and considerations that have to be respected in order to exploit the dynamic reconfigurability successfully. In addition, these ideas will provide a better understanding to the readers of the complexity of the dynamic reconfigurability in FPGAs.

Fig. 5. Cuprite image (250 × 191 pixels and 14 bands).
From the design point of view, a dynamically reconfigurable system has to split its design into two regions in the FPGA. On one hand, those parts of the system that never change have to be allocated in a static region, whereas the elements that are sensible to be modified during the execution have to be placed in a dynamic region. It is possible to create one or more dynamic regions into the same FPGA, depending on the system or the architectural needs. At the same time, the dynamic region is subdivided into smaller partitions known as dynamic areas. Within each of these dynamic areas a hardware component, with a specific functionality or a fake functionality, is deployed. In addition, the dynamic reconfigurability might be subdivided into two subcategories depending on how the dynamic region is affected during the reconfiguration process: the full dynamic reconfiguration, and the Dynamic and Partial Reconfiguration (DPR). In both cases, the static region keeps running during the reconfiguration process. Thus, the difference between them lies on the dynamic region status during the reconfiguration process. In a full dynamic reconfiguration process the whole dynamic region is configured. However, the DPR extends the inherent flexibility of the FPGA, since it only modifies specific elements in the dynamic region at run-time, without stopping the rest of modules of the device. Therefore, this feature fits perfectly with the demands of adaptable and scalable solutions.

One of the benefits motivated by the dynamic reconfigurability is the fact that the designer can dynamically insert new functionalities without redesigning the system or moving to a bigger device. Furthermore, it is possible to adapt the FPGA to different scenarios, by modifying the functionality or the performance of some tasks running on it. In addition, the DPR reduces costs and area due to it enables smaller designs by time-multiplexing portions of the available hardware resources. An example of this fact is presented in [13], in which a single hardware platform can support different waveforms using one or another at run-time according to the user/application requirements. The DPR feature provides additional advantages, such as reducing the bitstream storage needs, the synthesis time or accelerating the computation [14], [15].

In summary, the DPR process increases the powerful of the whole system by reusing elements that have been already implemented without redesigning the solution from the scratch, and after the system has been deployed. This characteristic is very convenient for all those applications that require real-time adaptability [16]. The DPR is a useful characteristic under those contexts in which the system cannot be stopped, but it needs to be adapted. Thus, the addition of new hardware components into the system requires that this one is instantiated in a partial reconfigurable region of the FPGA. Due to the fact that a dynamically reconfigurable FPGA might keep running, even when a component failure has been detected, it strengthens the robustness and the reliability of preemptive systems. Therefore, a system breakdown can be avoided by replacing the damaged component by a new one at run-time, improving in this way the system fault-tolerance [17].

Once the most important dynamic reconfigurability ideas have been introduced, the rest of this section goes in depth on how to apply these concepts onto a physical implementation. In this sense, Section IV-A describes how to manage the reconfiguration process independently from the processor, using a hardware element known as reconfiguration engine. Section IV-B is focused on controlling the resources allocation into the dynamic region. In this case, we have used a flexible methodology, different from the traditional ones, that makes the dynamic region more efficient in terms of the logic resources reusability. After these explanations, Section IV-C presents the use case (the MVCA application), and how one of the previous scalable

---

**Fig. 6.** Extracting 14 endmembers in a postplace and route behavioral simulation.

**TABLE IV**

<table>
<thead>
<tr>
<th>MVCA arch architecture</th>
<th>Logic resources’ utilization</th>
<th>Place and route report</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No. of slice registers</td>
<td>No. of slice LUTs</td>
<td>No. of DSPs</td>
</tr>
<tr>
<td>StaticArch_MVCA</td>
<td>35 372</td>
<td>26 164</td>
<td>54</td>
</tr>
<tr>
<td>SpectScalableArch_MVCA</td>
<td>36 960</td>
<td>27 915</td>
<td>54</td>
</tr>
<tr>
<td>SpatialScalableArch_MVCA</td>
<td>38 871</td>
<td>29 307</td>
<td>54</td>
</tr>
</tbody>
</table>
MVCA architectures has been modified in order to suit with the dynamic and partial reconfigurability constraints. Finally, Section IV-D exposes step by step the system behavior during the reconfiguration process.

A. Reconfiguration Engine

The dynamic reconfiguration management is traditionally afforded by using software approaches and a hardware component. However, the levels of parallelism are higher in hardware than in software since multiple tasks might be executed at the same time [18], whereas the software approaches follow sequential execution patterns. As a consequence, software solutions tend to be inefficient in terms of reconfiguration times rates, which are far from the technological limits. Most studies propose a software management of the partial reconfiguration process, in which a processor obtains a specific bitstream from a storage device, that can be later modified and sent to a reconfiguration controller [19], [20]. The strength of these alternatives lies on the flexibility provided by the software nature; unfortunately, the latency of the partial reconfiguration process is too high. As a possible solution to mask the latency, alternatives such as overlapping techniques between tasks planning and partial reconfiguration, or the prefetching of reconfigurable modules might be used. Nevertheless, these ones increase the complexity of tasks management [21].

The work presented in [20] is based in a fast dynamic reconfiguration process with little hardware requirements. Bitstreams are stored in an external memory, such as DDR, accessed from the processor through a Fast Simplex Link (FSL) [22]. Then the processor sends the data to the Internal Configuration Access Port (ICAP) core [23]. Despite the fact that this proposal uses a few resources, it takes a lot of time to reconfigure an area (about 25.89 MB/s). Therefore, in order to take advantage of the ICAP component clock frequency, it is necessary to reduce this time. On the other hand, [24] performs the reconfiguration process using a hardware component. It achieves a better reconfiguration time than the previous solution (about 33 MB/s). The drawback of both solutions is that the bandwidth used is limited by the bitstream rescue from the storage device. Accordingly, the ICAP bandwidth is not used completely, since it can work with 32-bit words, while these solutions use 8-bit words. This issue limits the reconfiguration time, which is increased considerably.

As an alternative, this work exploits the benefits of the hardware bitstream placement methodology presented in [25]. The management strategy relies on a specialized hardware component, called Reconfiguration Engine (prEngine from now on). This solution offers two main advantages compared to the software approaches. First, regarding the reconfiguration process itself, this proposal improves the reconfiguration speed in two orders of magnitude [25]. This is because hardware solutions get higher levels of parallelism. In this case, at the same time that a process is loading the partial bitstream from memory, another process is deploying the bitstream. Thus, the ICAP bandwidth is increased. Second, the prEngine is completely independent from the processor, since it controls the whole reconfiguration process on its own. In this sense, and with the aim of isolating the reconfiguration process from the technology and the device, the prEngine provides a common abstraction layer compatible with all the technologies, by means of encapsulating the mechanism as a transparent service to the upper abstraction layers.

More in detail, the prEngine might be seen as a hardware component constituted by several elements as shown in Fig. 7: the Factory, the Reconfiguration Controller (RController), and the iface. As a whole, it is responsible for offering a set of reconfiguration services through a simple bus interface, which is based on a bidirectional FSL and a unidirectional Native Port Interface (NPI) [26]. This mechanism makes the management of the reconfiguration process easier, caused by the fact that the service provided by the component is transparent to the user, who does not have to supervise reconfiguration process. The RController administrates the tasks related to the hardware reconfigurability and within its responsibilities are: the registration of all the hardware reconfigurable components (modules) before they can be instantated on the FPGA, the control of launching processes, etc. For these purposes, the RController includes a dynamic table (tableInst) that stores the instances location onto the reconfigurable region of the FPGA, their corresponding bitstream references, their execution status, and memory references for module persistence issues. It receives the reconfiguration requests from a higher entity (which can be a processor or any other software or hardware component). After that, it executes the corresponding operations on the demanded reconfiguration areas, or it might send a sequence of instructions to the Factory in order to fulfill with the request successfully.

The Factory component deals with the configuration bitstreams aspects, with the objective of adapting the information on-the-fly for relocating purposes, and the bitstreams transfers between memories. As its name suggests, it is able to create new instances from the modules, as an abstract factory pattern in software approaches [27]. Thus, the RController component sends a message to the Factory component, containing the identifier area and the address location of the module that will be deployed. This means, the location where the instance will be deployed and the reference where the module is stored. Then, the Factory component begins the transference
of the partial bitstream, matched with a module registered into the $RController$ tables, from the device memory to a temporal buffer, using a dedicated bus. In the particular case of the Xilinx FPGAs, the $Factory$ uses a NPI interface for moving data [26]. At the same time, the $Factory$ component reads the data from the temporal buffer and sends it to ICAP module. Hence, the $Factory$ component plays a role of a specialized $DMA$ controller. This process improves the throughput, being that the bandwidth increases, and at the same time the reconfiguration time decreases considerably.

The $prEngine$ communication is driven by the $Iface$ component by managing the messages that are sent by a client. These messages can be generated by different clients, such as a hardware controller, an embedded software, or even remotely through an Ethernet or serial link. Therefore, the $Iface$ component comes between the communication and the functionality of the $prEngine$ component, adding a new layer and supplying a transparent reconfiguration service.

### B. Reconfigurable Areas Layout

One of the main challenges associated to the dynamic reconfiguration lies on scaling the reconfigurable areas dynamically. The most conservative exploitation of the dynamic reconfiguration is based on dividing the reconfigurable region into predefined areas. Each one is determined by a fixed size, shape, placement on the reconfigurable region, and also a constrained interface with the rest of the areas and the static region. These characteristics are always predefined at design time and fixed at compilation time; consequently they cannot be modified according to the needs of the system at run-time. This tendency obligates to develop compatible components, in the case of being instantiated into the same reconfigurable area, in terms of interface, size and shape aspects. However, this approach is too restrictive when the reusability, the flexibility and the adaptability of all the elements of the system are critical. That is the case of systems that have to follow the variations of the environment, but keeping the system integrity by means of guaranteeing certain levels of performance and/or quality. Despite the fact that working on developing modular architectures incorporates higher levels of flexibility to a system; this characteristic is not enough for fulfilling the exigencies of external demands by itself. That is the case of some scenarios that require considering scaling its performance in order to attend the criticality of several tasks. Under these considerations, the performance scalability on hardware is afforded by modifying the number and nature of the running components dynamically, depending on the context [28]. In an attempt for solving these modern necessities, a hierarchical approach is necessary for organizing the reconfigurable region and its dynamic areas. This proposal is focused on reducing the constraints imposed over reconfigurable modules, since it allows for including a simple module or the combination of several ones at run-time into the same area. These two situations are graphically shown in Fig. 8, where one dynamic region is represented.

In Fig. 8(a), two independent areas have been created ($prApp1$ and $prApp2$) and both of them are communicated to the static region. Then, Fig. 8(b) represents a different situation, since the area on the left ($topApps$) is composed as a combination of two smaller areas ($prApp1$ and $prApp2$); whereas the third area ($prApp3$) is independent from the other. In return, the following considerations have to be contemplated.

1) In the case of operating with reconfigurable modules, both of them independent to each other, the composition of the reconfigurable region is the simplest case, due to the configuration is straightforward to the $prEngine$. Nevertheless, every reconfigurable module is restricted by the size and the shape of the reconfigurable region where the module will be deployed. Then, it is impossible to know beforehand the size of future reconfigurable modules [Fig. 8(a)]. The placement assignment can be easily solved using a grid approach [17], where a coarse-grain area (from the dimension and the number of resources point of view) might be subdivided into finer grain sub-areas inside.

2) In the case of requiring combining modules (like in the case of desiring scaling a solution), the interconnection between these modules brings up a new challenge, since there is to determine how and where to control the location of the shared signals between the modules. The proposed solution is based on the combination of several fixed signals together with a grid organization

![Fig. 8. Dynamic reconfigurable areas layout: a) Two independent modules ($prApp1$ and $prApp2$). b) Related modules ($prApp1$, $prApp2$, and $prApp3$).](image)
of the reconfigurable region layout, as it was previously described in [17]. Therefore, a fine grain subarea is defined with a fixed number of interconnection channels, which might be joined to the neighboring subarea in order to compose a coarser dynamic reconfigurable area. Another important aspect to consider at run-time is that those interconnection channels must be properly pulled up or down depending on whether they are used or not, respectively. The description of these channels and its nature has to be done at design time by the developer, since they can be defined as a bus or as independent signals of a protocol [Fig. 8(b)].

As a result, any module inside the grid may use one or more subareas for being instantiated. In addition, any combination of dependent or independent subareas is allowed, as far as the communication takes place using the predefined interconnection channels.

C. Dynamically Reconfigurable Architecture for the MVCA Application

In Section III-C, we concluded that the SpatialScalableArch\_MVCA architecture is the best candidate to suit with the demands of a variable environment, and at the same time to save resources depending on those environmental requirements. Although this architecture has been perfectly adapted to be scalable and also for accomplishing the dynamic reconfiguration process, this condition does not guarantee that the DPR might be performed with its current structure. If the system needs reusing the logic resources present into the reconfigurable region as much as possible (making space for other modules, functionalities, or modifying the performance), it is desirable that the idle resources remain available for running any other task at run-time. This adaptation will suppose a higher freedom to the system for organizing its resources and tasks, since the architecture will be able to be scaled on-the-fly, whereas other independent reconfigurable modules keep running without being interrupted. The architectural modification will achieve the improvement of some features such as the modularity, the flexibility, and the relocation of reconfigurable modules. An analysis of the constrains of the grid structure of the reconfigurable region, as they were explained in Section IV-B, makes easier to identify which parts of the scalable architecture have to be affected and also how. One of the main challenges to success is related to the placement and routing stage in the FPGA, since the communication lines between modules and between the static and the reconfigurable region have to remain fixed independently of the level of the scalability, or the number of modules allocated at run-time. Furthermore, due to the connectivity lines within an FPGA vary notably between horizontal and vertical resources, the designer/developer should ensure the shape of reconfigurable modules, and minimize the number of data wires attending to efficiency (area, power consumption, and clock frequency) principles. As a consequence, the SpatialScalableArch\_MVCA architecture needs including slight modifications into the design for supporting the dynamic and partial reconfigurability. In fact, the changes are focused on the signaling and data channels. They are grouped together under the same element, creating a regular communication element (the Dispenser). Internally, this element includes some basic control logic in order to distribute and send forward data between its corresponding neighboring.

Finally, in order to simplify the reconfiguration process in terms of time and management complexity, the PE and the Dispenser have been wrapped together as one element known as functional unit (FU), as shown in Fig. 9. Then, the MVCA scalability implies modifying a full FU by including or removing it from the reconfigurable region.

D. Dynamic Reconfiguration Process

The model for managing the reconfiguration process by using the prEngine is flexible enough for allowing the inclusion and the management of new components after the system deployment. The dynamic reconfiguration process occurs when one of the following situations happens: it is necessary to include a new functional component in the system, replace a faulty one, or to add an improved version of an already implemented component. First of all, it is important to highlight that every reconfigurable component has to be referenced univocally, in order to avoid collisions and misunderstanding during the reconfiguration process. The solution goes through assigning a unique identifier to every component, referred as Object Identifier (ObjectID). This code is formed by the corresponding module identifier (modID), and the instance identifier (instID). The former, modID, characterizes the module in terms of the object oriented paradigm and its object class.
The latter identifies to each one of the copies of the component that are already allocated in the reconfigurable region. In this sense, before adding new components in the system it is necessary to assign and register its new ObjectID, but also its corresponding memory address in the RController table. When the prEngine receives an invocation to instantiate a component into a specific dynamic area, the RController sends a message to the Factory. This message contains information for locating the component, such as the reference to the corresponding bitstream and the location of the dynamic area where the component is going to be instantiated. Then, the Factory loads the bitstream from memory and sends the data to the ICAP. After that the ICAP is responsible for loading the bitstream in the reconfigurable region. Once the Factory completes its tasks, it notifies the fact to the RController. Subsequently, the RController informs the client (the component that activated the process) by showing the prEngine status.

As it has been mentioned before, the configuration bitstreams play an important role within the dynamic reconfiguration process, since they contain the information for programming the logic of the FPGA. In particular, the dynamic scalability feature of the proposed solution requires using just one partial bitstream, which corresponds with the FU component. Thus, the starting configuration bitstream of the FPGA includes all the elements of the static region and the initial configuration of the reconfigurable region (in our case, including one FU), whereas a partial bitstream only has information of a particular reconfigurable component. In this sense, it is important to guarantee that the size, shape and connectivity characteristics of the FU unit match with the ones that will be instantiated in the scalable system. However, before storing this information, the header and the tail of the partial bitstream are deleted. This procedure removes the information related to the allocation of the element on the layout and reduces the amount of information to be stored. Then, the prEngine regenerates the header on-the-fly depending on the position of the required FU in the reconfigurable region.

V. Hardware Platform and Experimental Results

The solution described in this paper has been developed under a GNU/Linux environment, and has been implemented on a Xilinx FPGA Virtex5-LX110T platform. Fig. 10 represents the block diagram of the final architecture, which is divided into two parts: 1) the Static Part and 2) the Dynamic Part. Thus, Section V-A describes the Static Part and its components, considering the image reading process from a storage device as one of the most important tasks, since it is the main bottleneck of the system. The latter Section V-B presents the Dynamic Part of the hardware environment developed.

A. Static Part

The Static Part is divided in several components, some of them provided by Xilinx Inc., while other components have been developed ad hoc. More specifically, according to Fig. 10, it comprises four components: 1) a soft-core processor (a Microblaze), 2) a controller of the MVCA application (Controller MVCA), 3) a data transfer structure [composed by the set of a MultiPort Memory Controller (MPMC), the buffer-DDR and the pixelRescue modules], and 4) the prEngine (see Section IV-A). In this sense, the Microblaze microprocessor is responsible for running the software applications, like a reconfiguration scheduler. Despite the fact that it is not necessary to manage the reconfiguration process, the processor has been included in order to control the MVCA application, and report the results to a serial port. This serial port is managed by the RS232 component which shows the winner indexes obtained from the MVCA application on an emulated terminal. The MVCA application is divided in two parts: 1) the static process and 2) the dynamic one. The first one manages the static tasks of the MVCA application as was explained in section IV and matches with the MVCA Static Part component shown in the Fig. 10. On the other hand, the dynamic process is the Dynamic Part of the platform, where the set of used Functional Unit (FU) components will process the data. Being that the MVCA application has to read the pixels of the hyperspectral image from memory, this task is managed by two specific components (pixelRescue component and bufferDDR component) that will be explained along the next section. With the objective to being able to process hyperspectral images with different spatial sizes and number of spectral bands, the proposed system uses an external storage element (a DDR memory), instead of using the internal memory resources available in an FPGA (BRAMs blocks). The main reason behind this decision lies on the fact that number of BRAMs is limited, and in many times they are not enough for fulfilling the storage requirements for these kinds of images. The reading and writing operations in the DDR memory are supervised by the MPMC component [26], which implements a scheduling algorithm to attend all the data transfers performed through it.

In addition, the Controller MVCA component manages the communication between the MicroBlaze and the MVCA application, like a bus bridge, and it has been developed in hardware. Consequently, it is the responsible for the communication between the Dynamic and the Static Part of the platform. The configuration attributes, like the number of endmembers and FUs, are sent in a message by the software application through a FSL bus to the controller MVCA component. It translates the
message into specific signals to send them to the MVCA Static Part component. Besides, when the MVCA application extracts a winner index (a new endmember), the Controller MVCA component builds and sends a return message to inform the user about the result.

During the implementation stage, one of the challenges we had to face was the differences between the clock frequencies between the Static and the Dynamic Parts of the whole system. In fact, the Static Part operates with a clock frequency two times faster than the Dynamic Part (100 MHz). Actually, the pack composed by the Dynamic Part and the MVCA application operates at the same clock frequency (50 MHz) in order to keep the whole application synchronized during its execution. The main reason of this difference is due to physical layout issues, such as the vertical routing delays into the Dynamic Part for connecting the reconfigurable regions (pr1x_region).

Finally, the prEngine component manages the reconfiguration process onto the reconfigurable region, according with the system requirements. Its structure and functionality was explained in Sections IV-A and IV-D, respectively.

1) Reading a Hyperspectral Image From an External DDR Memory: Data Transfer Structure: One of the main bottlenecks of this hardware platform is the process of getting pixels from the external storage device, in this case the DDR memory. Despite the fact that these memories are bigger than the BRAMs, their reading and writing operations are slower than using BRAMs. This inconvenient makes the system slower and degrades its overall performance. In order to alleviate this issue, we propose a solution that improves the memory access time and also increases the level of parallelism.

As it was previously mentioned, the memory transfers are managed by the MPMC component, which acts as a multiplexor. Thus, it is responsible for deciding which component might read or write from/to memory and when. In the proposed scenario, the memory accesses are unidirectional, since only reading operations are required from the MVCA application. The communication from the MPMC component to other components is driven by a NPI bus, which allows getting a set of pixels in only one transfer [26]. Thus, prEngine, the bufferDDR and the pixelRescue components are connected with the MPMC through a NPI bus each one.

More in detail, the bufferDDR and the pixelRescue components have different purposes in the system, and they are used at different moments according to the MVCA application needs. In this way, the former is used for calculating the pixels projections when the IMAGE PROJECTION module is processing. However, the pixelRescue component is required once the MVCA application gets the index of a winner endmember. Thus, when the MVCA application completes an endmember extraction, it requests the pixel from a specific memory address to the pixelRescue component, which gets the pixel stored into the DDR memory. These endmembers are internally stored into the pixelRescue component, and are used by the MVCA applications when corresponds. Therefore, the number of 32-bit memory addresses depends on the number of endmembers of the image. In the case of the Cuprite image (Fig. 5), the endmember extraction process will complete 14 memory addresses at the end of the computation.

Fig. 11. Block diagram of the bufferDDR component.

Fig. 12. Dynamic Part layout of the developed embedded system, with three reconfigurable areas (pr1l_region, pr12_region, and pr13_region).

Regarding the bufferDDR component, this is an intermediate buffer between the DDR memory and MVCA application as shown in Fig. 11. First, the scheduler of the system has to configure the memory address where the image is stored and its size in pixels; the admin module stores these attributes and activates a flag to begin the reading process. Then, the bufferDDR component reads the image data from the DDR memory through a NPI bus connected to the MPMC component, and stores these data in an internal FIFO (FIFO IN). At the same time, the pixel adapter module reads the data stored in that FIFO and builds the pixel with its endmembers. After the pixel is built, it is stored in another FIFO (FIFO OUT). Once there are enough pixels, the admin module informs to scheduler that it is possible to start the MVCA application. After that, the MVCA application does some requests to the bufferDDR component for reading the image pixels. Due to the behavior of the MVCA application, in which the hyperspectral image needs to be read as many times
as the number of endmembers to extract, the bufferDDR component acts as a circular buffer. This means that it begins to read the image from the beginning after reading the last pixel.

**B. Dynamic Part**

The Dynamic Part is structured into three reconfigurable areas (pri1_region) as shown in Fig. 10, and it is in charge of managing the dynamic tasks of the MVCA application. Every reconfigurable area might contain a component according to the user requirements, like a functional unit (FU) or a cover component. Thus, the logic resources might be reused for performing different purposes in different ways. This feature improves the performance of the MVCA application by allowing to scale the number of FUs, from one to three, present into the Dynamic Part at run-time. The intermediate registers between every reconfigurable region facilitates, in this particular case, to keep the synchronization and the clock frequency of the system.

An example is shown in Fig. 12, the areas called pri1_region and pri2_region contain a FU instance each one, where the busy resources are highlighted, while the pri3_region is completely empty due to the fact that it carries out the cover role. If the software application requires a new FU component, the prEngine component will deploy this new FU in the pri3_region. On the other hand, if the software application is able to run with a low performance, the prEngine component will deploy a cover component in the pri1_region region.

**C. Experimental Results**

This section collects the results obtained from the deployment of the embedded system described in the last section, which has been implemented on a Xilinx Virtex5-LX110T FPGA as shown in Fig. 10. The purpose of the system is the extraction of the 14 endmembers of the selected subset of Cuprite image that has been stored in an external DDR memory, and simultaneously trying to fulfill with real-time constraints. The scalability property of the implemented design permits to adapt the execution time of the MVCA algorithm according to the number of FU components dedicated to compute data. Consequently, the execution time required to obtain an end-member is different when the platform has one, two or three FU components, being the less favorable case when there is only one FU deployed in the system. This fact is corroborated through the information collected in Table V, in which the logic resources’ utilization and the performance results are shown when the three possible scenarios are considered: one, two and three FUs deployed. Despite the fact that this dynamic and partial reconfigurable MVCA design uses up to three FUs and stores the Cuprite image in an external DDR memory, its execution time improves compared with the simulation’s results of the SpatialScalableArch_MVCA design with 14 FUs. These execution enhancements are based on design modifications applied on the SpatialScalableArch_MVCA approach for moving from a scalable solution toward a dynamic and partially reconfigurable one. However, these execution enhancements have also impacted in the required area, since the number of hardware resources (into the reconfigurable region) has been enlarged according to the three possible scenarios.

Another important aspect of this system is the mechanism for managing the dynamic and partial reconfiguration process. That is, the prEngine component. The time required for switching the contexts of the system, according to the required level of scalability, is a critical aspect to be considered. In fact, this interval of time should be shorter and much lower than the execution time in order to provide an efficient and fast reconfiguration process. In this sense, the prEngine manages the inclusion and removal of the FU components in specific reconfigurable areas of the Dynamic Part of the system. More specifically, this component is able to achieve a bitrate up to 180 MB/s, which improves the reconfiguration time results of other works [20], [24]. This improvement is due to the use of the full bandwidth of the ICAP, and the parallel execution of the configuration management tasks. The cost of this speedup is an increment into the number of hardware resources compared to other solutions, such as Table VI shows.

In regards to the Dynamic Part, the three dynamic areas have the same size, shape and number of available resources, although the final occupancy depends on the characteristics of the deployed component on them (Table VII).
Finally, the reconfiguration time for including a new element into the Dynamic Part of the system depends on the two previous concepts: the bitrate of the prEngine component and the characteristic of the module to be deployed. Thus, Table VIII includes the reconfiguration time for reconfiguring the system with one, two or three FUs respectively. These numbers make sense, since the bitstreams size increases with the increment of the logic resources that have to be modified during the reconfiguration process, and as a result, the reconfiguration time is also higher.

### VI. Conclusion

This paper presents a dynamic and partially reconfigurable embedded system for hyperspectral image processing that allows extracting the endmembers of a hyperspectral image onto a Xilinx Virtex5-LX110T FPGA according to the MVCA algorithm. In this solution, the data transfer mechanism between the storage device (an external DDR memory) and the system itself plays a critical role. Thus, an inappropriate control of these transfers might suppose a bottleneck in the system, since the MVCA is very intensive in data and requires many memory accesses in order to perform its functionality properly. Consequently, the final performance will directly depend on the control of these data movements. Apart from this aspect, the efficiency of the proposed system also depends on the following factors: the reconfiguration time, and the hardware scalability feature’s exploitation.

Regarding the reconfiguration time, the proposed embedded system incorporates a reconfiguration engine component (prEngine) capable for accelerating the dynamic and partial reconfiguration process by getting a bit rate up to 180 MB/s. The couple conformed by this component, together with the flexible strategy that has been adopted for adapting the reconfigurable areas of the Dynamic Part of the system to the scalability, has permitted to reach various objectives. First, the system might modify its internal structure by deploying new FU components at run-time without resynthezing the system. Even more, the application might be scaled by replicating the computation element (FU) into the reconfigurable region. Second, the proposed system is able to compute the MVCA task fulfilling with real-time constraints, according to the AVIRIS rates. It should be noted that the cross-track line scan time in AVIRIS is quite fast (8.3 ms to collect 512 full pixel vectors). This introduces the need to process an AVIRIS Cuprite scene of 350 × 350 pixels and 244 bands in less than 1.985 s to fully achieve real-time performance. Therefore, in our case, the Cuprite scene of 250 × 191 pixels and 14 bands should be processed in less than 56.835 ms. This value is higher than the execution time of our system, even in the case of operating with only one FU. As a consequence, the proposed embedded system is suitable for real-time hyperspectral endmember extraction.

Finally, the proposed embedded system provides a scalable solution meeting the requirements of the MVCA application, which might be dynamic and partially reconfigured by a hardware component that reduces the reconfiguration time against other proposals, as the experimental results have demonstrated.

### References


---

**Table VII**

<table>
<thead>
<tr>
<th>Area</th>
<th>No. of LUTs</th>
<th>No. of slices</th>
<th>No. of FLP/FLOPS</th>
<th>No. of DSPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Available</td>
<td>728</td>
<td>182</td>
<td>728</td>
<td>4</td>
</tr>
<tr>
<td>FU component</td>
<td>116</td>
<td>56</td>
<td>180</td>
<td>4</td>
</tr>
</tbody>
</table>

**Table VIII**

<table>
<thead>
<tr>
<th>Components</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 FU component</td>
<td>249.15</td>
</tr>
<tr>
<td>2 FU components</td>
<td>487.47</td>
</tr>
<tr>
<td>3 FU components</td>
<td>731.20</td>
</tr>
</tbody>
</table>
[27] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Reading, MA, USA: Addison-Wesley, 1995, pp. 99–110.

Teresa G. Cervero was born in Asturias, Spain, in 1982. She received the M.Sc. degrees in telecommunications engineering and advanced telecommunications engineering in 2007 and 2007, respectively, and the Ph.D. degree in 2013, all from the University of Las Palmas de Gran Canaria (ULPGC). She got the Ph.D. degree also by the ULPGC in 2013, where she currently collaborates as a Member of the Integrated Systems Design Division (DSII), Institute for Applied Microelectronics (IUMA), University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain.

Her research interests include real-time designs for multimedia applications, H.264/AVC and SVC video codecs, hyperspectral imaging systems, synthesis-based design for SoCs, and reconfigurable computing.

Julian Caba received the B.S. and M.S. degrees in computer science from the University of Castilla-La Mancha, Ciudad Real, Spain, in 2006 and 2009, respectively, where she is currently pursuing the Ph.D. degree at the Department of Technologies and Information Systems, University of Castilla-La Mancha, Ciudad Real, Spain.

Her research interests include hardware verification methodologies, high-level synthesis, reconfigurable computing, and heterogeneous distributed systems.

Sebastián López (M’08) was born in Las Palmas de Gran Canaria, Spain, in 1978. He received the degree in electronic engineering from the University of La Laguna, in 2001, obtaining regional and national awards for his CV during his degree, and the Ph.D. degree from the University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, in 2006. He is currently an Associate Professor with the University of Las Palmas de Gran Canaria, developing his research activities at the Integrated Systems Design Division of the Institute for Applied Microelectronics (IUMA). He has published more than 60 papers in international journals and conferences. His research interests include real-time hyperspectral imaging systems, reconfigurable architectures, video coding standards, and hyperspectral compression systems.

He is a member of the IEEE Geoscience and Remote Sensing and Consumer Electronics Societies as well as an Associate Editor of the IEEE TRANSACTIONS ON CONSUMER ELECTRONICS. Additionally, he currently serves as an active reviewer of the IEEE JOURNAL OF SELECTED TOPICS IN APPLIED EARTH OBSERVATIONS AND REMOTE SENSING (JSTARS), the IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING, the IEEE GEOSCIENCE AND REMOTE SENSING LETTERS, the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, the Journal of Real Time Image Processing, Microprocessors and Microsystems, Embedded Hardware Design (MICPRO), and the IET Electronics Letters, among others. He is also a Program Committee Member of different international conferences including the SPIE Conference on Satellite Data Compression, Communication and Processing, IEEE Workshop on Hyperspectral Image and Signal Processing: Evolution in Remote Sensing (WHISPERS), and SPIE Conference of High Performance Computing in Remote Sensing. Furthermore, he has been designated as the program Co-Chair of the last two aforementioned conferences for their 2014 editions.

Julio Daniel Dondo completed the graduate studies in electrical-electronic engineering and received the M.Sc. degree in software engineering from the National University of San Luis, San Luis, Argentina, in 1996 and 2007, respectively, and the Ph.D. degree from the University of Castilla-La Mancha, Ciudad Real, Spain, in 2010.

He is currently working as an Associate Professor with the University of Castilla-La Mancha. His research interests include the integration of complex embedded systems, reconfigurable computing, heterogeneous distributed systems and Reconfigurable Grid Computing.

Roberto Sarmiento received the Ph.D. degree in electronic engineering from the University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, in 1991.

He is currently a Full-Professor of Electronic Engineering with the Telecommunication Engineering Faculty, University of Las Palmas de Gran Canaria (ULPGC), Las Palmas de Gran Canaria, Spain. He was the Dean of the Faculty from 1994 to 1998 and Vice-Chancellor for Academic Affairs and Staff with the ULPGC from 1998 to 2003. He was a Visiting Professor with the University of Adelaide, South Australia, in 1993, and later with the University of Edith Cowan, Joondalup, Australia. He is a Co founder of the Institute for Applied Microelectronics (IUMA) and Director of the Integrated Systems Design Division of this Institute. Since 1990, he has published more than 40 journal papers and book chapters and more than 120 conference papers. He has participated in more than 35 projects and research programs funded by public and private organizations, from which he has been lead researcher in 16 of them. He has conducted several agreements with companies for the design of high performance integrated circuits, being the most remarkable the collaboration with Vitesse Semiconductor Corporation, Camarillo, CA, USA and Thales Alenia Space, Madrid, Spain. His research interests include multimedia processing and video coding standard systems, reconfigurable architectures and real-time processing, and compression of hyperspectral imaging.

Prof. Sarmiento has been awarded with four six years research periods by the National Agency for the Research Activity Evaluation in Spain.
Fernando Rincón (M’98) completed the graduate studies in computer science at the Autonomous University of Barcelona, Barcelona, Spain, in 1993 and received the Ph.D. degree from the University of Castilla-La Mancha, Ciudad Real, Spain. He is currently an Assistant Professor and Head with the TSI Department, University of Castilla-La Mancha. His research interests include System-On-Chip integration, HW run-time reconfiguration, and Heterogeneous Distributed Systems. Prof. Rincón is a member of the ACM.

Juan Carlos López (M’92) received the M.S. and Ph.D. degrees in telecommunication (electrical) engineering from the Technical University of Madrid, Madrid, Spain, in 1985 and 1989, respectively. From September 1990 to August 1992, he was a Visiting Scientist with the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA. From 1989 to 1999, he was an Associate Professor with the Department of Electrical Engineering, Technical University of Madrid, Madrid, Spain. In 1999, he joined as a Professor of Computer Architecture with the University of Castilla-La Mancha, Ciudad Real, Spain, where he served as Dean of the School of Computer Science from 2000 to 2008. Since 2006, he is the Director of the Indra Chair with the same university. His research activities include embedded system design, distributed computing, and advanced communication services. He is and has been a member of different panels of the Spanish National Science Foundation and the Spanish Ministry of Education and Science, regarding the Information Technologies research programs. Prof. López is a member of the ACM.