Run-time adaptable FPGA-based embedded system for real-time hyperspectral unmixing

Conference Paper · November 2014

CITATIONS
0

READS
23

1 author:

Julian Caba
University of Castilla-La Mancha
12 PUBLICATIONS **11** CITATIONS

SEE PROFILE

Run-time adaptable FPGA-based embedded system for real-time hyperspectral unmixing

J. Caba, J.D. Dondo, F. Rincón, J.C. López
Department of Information Technologies and Systems
School of Computer Science, UCLM
Ciudad Real, Spain
[julian.caba, juliodaniel.dondo, fernando.rincon, juancarlos.lopez]@uclm.es

T. Cervero, S. López, R. Sarmiento
Institute for Applied Microelectronics
IUMA, ULPGC
Las Palmas de Gran Canaria, Spain
[tcervero, seblopez, roberto]@iuma.ulpgc.es

***Abstract*—Hyperspectral imaging instruments capture and collect hundreds of different wavelength data corresponding to the same surface. One of the requirements of paramount importance, when dealing with applications that demand swift responses, is the ability to achieve a high real-time performance. In this sense, the authors present a flexible and adaptable 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 data parallelism by scaling its execution in hardware. The solution uses the dynamic and partial reconfiguration property of FPGAs together with a hardware reconfiguration engine in order to exploit and vary the number of processing elements at run-time.**

Keywords—Dynamic and partial reconfiguration, FPGA, hyperspectral imaging, linear unmixing, scalability.

I. INTRODUCTION

Hyperspectral imaging is becoming 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. The lack of spatial resolution provided by the hyperspectral imaging instruments, such as the Airbone Visible Infra-Red Imaging Spectrometer (AVIRIS) [1], makes that each pixel being viewed as a mixture of several spectrally pure distinct materials. There exist several techniques for analyzing this kind of images, but the most extended one is based on linear unmixing models. Thus, each pixel might be characterized as a linear combination of pure materials present in the scene, but contributing in different percentage.

Within the spectra of linear unmixing algorithms, the Vertex Component Analysis (VCA) [2] is one of the most successful and popular algorithm. Thought it has better results than others, its implementation is complex because of its high computational requirements, and the nature of its operations. As an alternative, the Modified Vertex Component Analysis (MVCA) algorithm, which has been developed by López et al. [3], has been demonstrated to outperform the VCA algorithm in terms of endmember extraction accuracy vs computational complexity. Beyond traditional software implementations, the MVCA still needs to be further accelerated in order to accomplish this with real-time or near real-time applications requirements. Thus, Field Programmable Gate Arrays (FPGAs) stand out as an adequate candidate for dealing with the MVCA needs. Their success lies on their smaller size and weight compared with traditional cluster-based systems, as well as to their lower power dissipation figures compared to the Graphics Processing Units (GPUs) [4]. Moreover, the low data dependences, the regularity and the huge amount of data of the MVCA make it ideal to widely exploit the advantages of the parallelization offered by the FPGAs.

Furthermore, the run-time reconfigurability feature offered by FPGAs opens up a wide range of possibilities to deploy a new generation of adaptable and scalable solutions capable of coping with the stringent demands of real-time hyperspectral imaging applications. Hence, the exploitation of the scalability at run-time allows the system to be adapted to the image characteristics, or to other external demands, by modifying the number of resources involved in the computation. However, these variations bring some consequences. One of them is the development of a flexible parallelization strategy capable for adapting the computation according to the system requirements. Also, an accurate and efficient management of these variations at run-time plays an important role in this context. In the end, these facts as a whole make possible saving area, resources, power consumption, increasing the reusability of idle resources, among others. With these considerations, this paper presents a scalable and dynamically reconfigurable FPGA-based solution for the endmember extraction MVCA algorithm. The solution might vary and efficiently manage the level of parallelism by modifying the number of processing elements involved in the execution at run-time. In addition, the proposed approach also explores a new paradigm on the utilization of the logic resources within the reconfigurable area.

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

This work has been supported by the Spanish Government within the R&D DREAMS Project (TEC2011-28666-C04-03).

platform and evaluates the implementation performance. Finally, Section VI outlines the conclusions of the paper.

II. THE VCA AND MVCA ENDMEMBER EXTRACTION ALGORITHMS

The linear mixture model considers that these mixed pixels are a linear combination of the pure pixels (endmember) signatures present in the scene weighted by the correspondent abundance fractions (i.e., the percentage of each endmember). This model considers the hyperspectral image as a matrix of N bands and R pixels ($\mathbf{Y} \in \mathbb{R}^{N \times R}$). Then, each pixel can be obtained as:

$$\mathbf{r} = \Sigma\, a_i \cdot e_i + n;\ \{ i=1..p\} \qquad (1)$$

where each pixel ($r = [r_1, r_2, \ldots, r_N]^R$) can be represented as a linear combination of a finite set of endmembers (e_i), weighted by an abundance factor (a_i), and **n** is an additive noise vector. The total number of endmembers in an image is p. In general, the procedure of linearly unmixing is described by three main stages. First, the estimation of the number of endmembers (p); then, the extraction of their spectral signatures ($\mathbf{E}=[e_1, e_2,..e_p]$); and finally, the estimation of the respective abundance fractions ($\mathbf{a}=[a_1, a_2\ldots, a_p]$) for each pixel.

During the last decade, a huge amount of algorithms have been proposed for each of these three stages. In this sense, interested readers can find an excellent and comprehensive review of all of them in [5]. A common drawback of these algorithms is that their computational complexity prevents from their use in applications that require a swift response. In order to alleviate this problem, this work is focused on the acceleration of the second of the aforementioned stages, i.e. the endmember extraction. Although there exist a wide collection of endmember extraction algorithms, this work is focused on the MVCA. This algorithm is able to obtain the same level of performance than the VCA but with simpler operations, which reduces the computational time to obtain the endmembers.

A. VCA algorithm

VCA algorithm is based on the algebraic fact that the endmembers are the vertices of a simplex (positive cone defined by the hyperspectral data to be processed), being the affine transformation of a simple also a simplex. The projection of that cone into a properly chosen hyperplane gives a simplex, where its vertices correspond 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. Then, the VCA iteratively projects data onto a direction orthonormal (given by a vector named *f*) to the subspace spanned by the endmembers already determined. The number of endmembers to be extracted is defined by p, which is known in advance.

Mathematically, the VCA algorithm is composed of a set of operations where can distinguish at least two computational processes with a high degree of complexity. The first one is related to the generation of vector *f*, which is defined as an orthonormal vector to the already calculated endmembers. This process requires computing the pseudoinverse of the matrix where the already computed endmembers are stored. The second more complex operation is the projection of the hyperspectral data onto the direction pointed by vector *f*. Since the number of pixels in a hyperspectral image is usually large, the amount of floating point operations per second (flops) is usually huge per iteration at the VCA algorithm.

B. MVCA algorithm

The MVCA algorithm is able to reproduce the results of its original counterpart in terms of endmember extraction accuracy, while exhibiting a much lower computational complexity. Actually, the MVCA algorithm outperforms the VCA by applying a low-complexity orthogonalization method and by utilizing integer instead of floating–point arithmetic when dealing with hyperspectral data.

This performance is achieved by introducing two main modifications into the processes by which vector f is calculated. First, the norm of this vector is not forced to be equal to the unity, which means that the vector f is not normalized. Secondly, 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 is changed. 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 i 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:

$$u_k = e_k - \Sigma\, proj\,(e_k,u_j);\quad \{j=1..k-1;\ k= 1..\ i \text{ and } u_1=e_1\} \qquad (2)$$

Where the *proj* stands for the projection operator, defined as:

$$proj\,(e_k, u_j) = u_j \cdot \langle e_k,u_j\rangle / \langle u_j, u_j\rangle;\quad \text{where } \langle x,y\rangle = \Sigma\,(x_z \cdot y_z)\ \ \{z= 1..p\} \qquad (3)$$

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 pseudo-inverse of matrix E. Vector u_{i+1} is computed by the following expression:

$$f = w - \Sigma\, proj\,(w,u_l);\ \{l = 1\ ..\ i\} \qquad (4)$$

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, where the code is executed sequentially. In an attempt to improve the MVCA performance, this section explores how to accelerate the MVCA execution in hardware by using two different and flexible parallelization strategies onto reconfigurable FPGAs. One of them exploits a spectral parallelization technique, whereas the other uses a spatial one.

The reference hardware design that we have used in this work is detailed in [3], and represented in Fig. 1. Here, the MVCA is structured in modules, in which every element has a specific task.

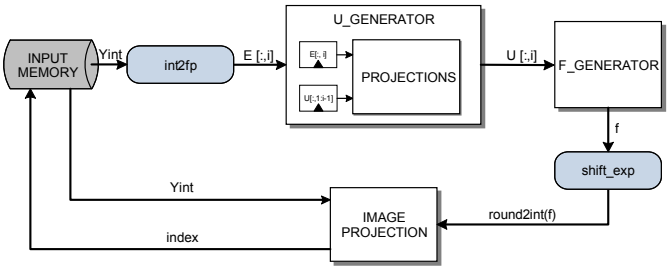

Fig. 1. Block diagram of the MVCA hardware design

According to Fig. 1, the computation modules are the *u_generator*, the *f_generator* and the *image projection*. The *input memory* module stores the hyperspectral image as a matrix, and the rest of modules are responsible for data format conversions.

- *U_generator* generates a set of vectors (*U*) orthogonal to the endmembers that have been already calculated (*E*) by using a Gram-Schmidt orthogonalization method.

- *F_generator* produces the vector *f* as an orthogonal vector of the set of *U* vectors.

- *Image projection* calculates the projection of all the pixels that compose the hyperspectral image onto the direction pointed by the vector *f*. This operation is based on a matrix multiplication. As a result, it produces an index that corresponds with the memory address of the endmember. Due to the regularity of its operations, the low data dependences and the large amount of information to be processed, the *image projection* is the best candidate to be parallelized. All these characteristics facilitate the exploitation of the data level parallelism (DLP), providing a high flexibility to perform more than one operation simultaneously.

Despite the fact that there exist parallel solutions in hardware, this paper introduces a higher flexibility by scaling the solution in hardware. In fact, the scalability is exploited by varying the number of processing elements (PEs) involved in the computation of the *image projection* module.

Before going in detail on the proposed parallel MVCA solutions, it is necessary to remark that the scalability only has been applied on the *image projection* module, and hence the rest of functional modules are the same for both approaches: the spectral and the spatial.

A. Spectral vs spatial scalability

The spectral parallel solution processes one pixel projection at a time, and its scalability impacts on the number of spectral components that are computed simultaneously. On the contrary, the spatial parallel approach scales the number of pixels' projections that are concurrently computed. Despite the differences between these two approaches, they share two characteristics in common. On the one hand, the computation is performed by the same kind of processing element (PE), formed by the combination of a multiplier and an accumulator (MAC). On the other hand, the scalability level might vary between one and *p*.

1) Spectral scalability: SpectScalableArch_MVCA

Depending on the number of PEs, this approach computes one or several spectral components of the same pixel simultaneously. All the PEs start processing at the same time, and once all the spectral components have been computed, the pixel projection is calculated by adding all the results by an adder-tree module. Then, the projection is stored only in case its value is higher than the highest value calculated until that moment. The schematic of this solution is represented in Fig. 2.

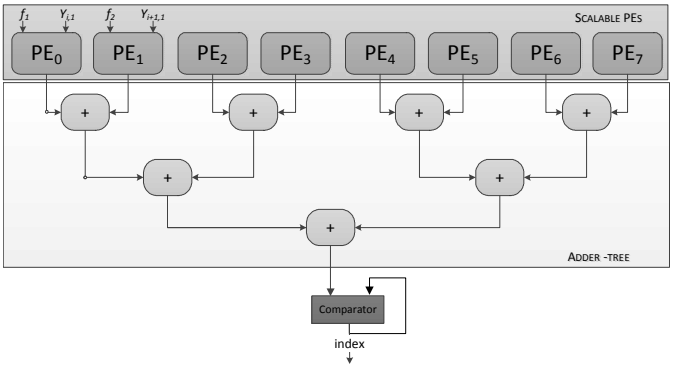

Fig. 2. Block diagram of the *SpectScalableArch_MVCA* hardware design

The possibility of modifying the number of PEs introduces flexibility, hardware reusability and adaptability to the system. As a result the execution time for extracting the endmembers might be adapted to the system or the environmental conditions. However, when a pixel projection computation requires the same number of clock cycles ($p/2<nPE<p$), a high scalability does not always implies a performance improvement. Furthermore, when $nPE = 1$ a pixel projection is completely performed by the same PE, and consequently the adder-tree remains idle during the process, but occupying resources, since it belongs to the static part of the design.

2) Spatial scalability: SpatialScalableArch_MVCA

The spatial parallelization dedicates every PE to complete a pixel projection. In this way, in case of having more than one PE, several pixel projections might be computed simultaneously. One of the strengths of this strategy is the fact that the adder-tree is not necessary, since the PE accumulates all the intermediate results and its final result corresponds to the pixel projection itself. The schematic of this solution is depicted in Fig. 3.

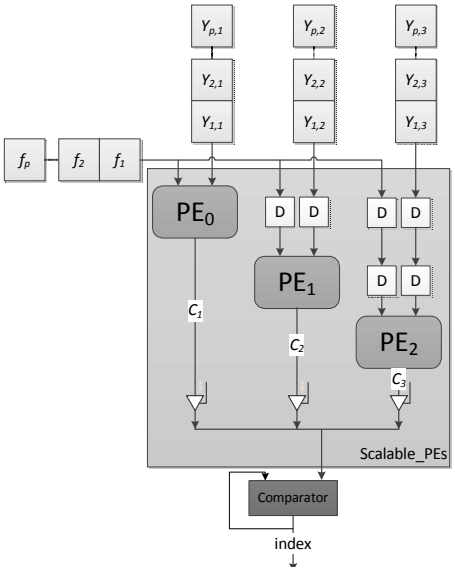

Fig. 3. Block diagram of the *SpatialScalableArch_MVCA* hardware design

This design is reduced to two blocks, the *Scalable_PEs* and the *comparator*. This fact has been possible by delaying one clock cycle the start processing of consecutive PEs. In this way, and by limiting the scalability level between one and *p*, it is possible to guarantee that, in the worst case, only one pixel projection is generated per clock cycle.

Definitively, the benefits of deploying scalable approaches are reached at cost of using more hardware resources than traditional parallel and static solutions (when using the higher scalability level). After the implementation of these two approaches (the *SpectScalableArch_MVCA* and the *SpatialScalableArch_MVCA*) on a Xilinx Virtex-5 LX110T FPGA, the reported clock frequencies were 109,445MHz and 149.70 MHz, respectively. The number of resources is higher using the spatial parallelization than in the spectral one (5.7%. considering the slice registers, and 4.98% considering the number of slice LUTs). Finally, the *SpatialScalableArch_MVCA* approach has no limitations regarding the execution time when the scalability level increases, even when it is between *p/2* and *p*.

B. Scalable and dynamic and partially reconfigurable MVCA solution

The fact of developing a scalable solution does not imply having a dynamically and partially reconfigurable (DPR) design. In this sense, this section details how to adapt the *SpatialScalableArch_MVCA* design in order to fulfill with the DPR constraints. This adaptation will maximize the reusability of the unused resources, since it supposes 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 be interrupted. The architectural modifications are focused on improving the relocation capability of the reconfigurable modules, and also on reducing the impact of the physical limitations of an FPGA. As a consequence, the modifications are based on making the signaling and data channels regular, and reducing the number of data/control lines as much as possible. In this case, it is possible to reduce the vector *f* lines, since its components might be sent forward between consecutive PEs. Regarding the regularity, it is important to create a new regular and homogeneous element in charge of connecting and feeding with data the PEs. As a result, all the data and control channels related to a PE are collected under one regular element (the *Dispenser*). The PE does not change. However, 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). Therefore, increasing the scalability level implies the inclusion of one or several FUs, instead of only one PE.

IV. DYNAMIC RECONFIGURABILITY

From the design point of view, a dynamically reconfigurable system distinguishes between the static and the reconfigurable regions in the FPGA. The static region contains all those parts of the system that never change during the execution. On the contrary, all those elements that are susceptible to be modified during the execution must be allocated within the reconfigurable region. At the same time, the reconfigurable region might be subdivided into smaller partitions known as dynamic areas. Further from the strengths of using DPR, its use is not widely spread due to the complexity of its management. However, following subsections explain in detail how we have solved this challenge.

A. Reconfiguration Engine

This work exploits the benefits of the hardware bitstream placement methodology presented in [6]. 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. Firstly, this proposal improves the reconfiguration speed in two orders of magnitude. In fact, it allows a process to load the partial bitstream from memory while another process is deploying the bitstream. Secondly, the *prEngine* is completely independent from the processor, since it controls the whole reconfiguration process on its own.

The *prEngine* is responsible for offering a set of reconfiguration services, and for that it is constituted by several elements as Fig. 4 depicts: the *Factory*, the *Reconfiguration Controller (RController)* and the *Iface*. The *RController* administrates tasks related to the hardware reconfigurability, such as the registration of the hardware reconfigurable components before they can be instantiated on the FPGA, the control of launching processes, etc.

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 transferences between memories. As its name suggests, it is able to create new instances from the modules, as an *abstract factory pattern* in software approaches [7]. This process improves the throughput, being that the bandwidth increases, and at the same time the reconfiguration time decreases considerably.

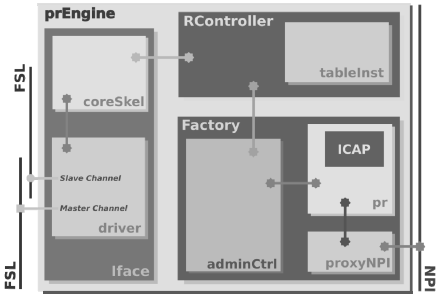

Fig. 4. Block diagram of the MVCA hardware design

The *Iface* component manages the messages that are sent by a client to the *prEngine*. Therefore, it 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

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 with those constrains.

However, this approach is too restrictive when the reusability, the flexibility and the adaptability of all the elements of the system are critical. With the objective of supporting these features, the authors present a hierarchical approach capable for organizing the reconfigurable region and its dynamic areas by reducing the constraints imposed over reconfigurable modules. This hierarchical approach allows including a simple module or the combination of several ones at run-time into the same area, as it shown in Fig. 5. In addition, any combination of dependent or independent sub-areas is allowed, as far as the communication takes place using the predefined interconnection channels.

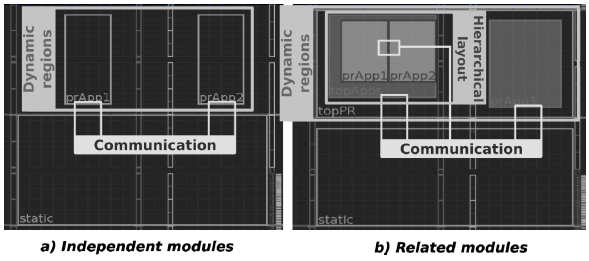

Fig. 5. Block diagram of the MVCA hardware design

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. The Fig. 6 represents the block diagram of the final architecture, which is divided in two parts: the *Static Part* and *Dynamic Part.*

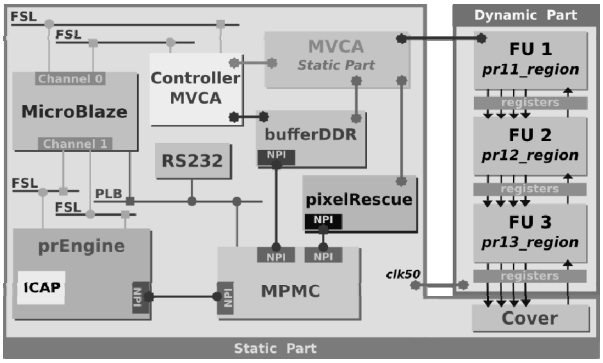

Fig. 6. Block diagram of the MVCA hardware design

The application that has been implemented is the DPR MVCA application, which was explained in Section III.B. Due to the hardware resources of the FPGA, the scalability level is limited between one and three FUs. This scalability depends of the application requirements, thus if the applications does not require a high performance, the MVCA application will use a dynamic area, whereas if the application needs a high performance, the final platform will use several FUs, in this case two or three FUs.

A. Static Part

The *Static Part* is comprises four main components: a soft-core processor (a *Microblaze*), a controller of the MVCA application (*Controller MVCA*), a data transfer structure (composed by the set of a Multi-Port Memory Controller (*MPMC*), the *bufferDDR* and the *pixelRescue* modules), and the *prEngine*. In this sense, the *Microblaze* microprocessor is responsible for running the software applications, like a reconfiguration scheduler. Due to there is only one application running in the system, 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.

B. Reconfigurable Part

The *Dynamic Part* is structured into three reconfigurable areas (*pr1x_region*) as Fig. 6 shows, 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 the number of FUs to be scale, from one to three at run-time.

C. Results

The purpose of the system is the endmembers extraction of the 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.

During the implementation stage, one of the challenges we faced was the differences between the clock frequencies between the *Static* and the *Reconfigurable Parts*. While the former is able to process at 100 MHz, the latter only reaches 50 MHz. This difference is due to physical layout issues, such as the vertical routing delays into the *Reconfigurable Part* for connecting the reconfigurable regions (*pr1x_region*). TABLE I. collects information related to the resources utilization of the whole system, considering the place and route information report, when different scalability levels are enabled.

TABLE I. PLACE AND ROUTE REPORT OF THE DPR FPGA-BASED MVCA SYSTEM

FUS	SLICE REGS.	SLICE LUTS	DSPS	FREQ. (MHZ)
1	44,443	36,821	19	100.702
2	45,226	37,414	22	100.702
3	45,951	38,004	25	100.702

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 *Reconfigurable Region* of the system. More specifically, this component is able to achieve a bitrate up to 180MB/s, as TABLE II. shows.

TABLE II. RECONFIGURATION ENGINE COMPARISON

SOLUTION	SLICES	LUTS	FLIPFLOPS	RATE (MBYTES/S)
XILINX [8]	616	1,405	728	14.5
HUBNER [9]	78	44	10	25.89
XCELL [10]	2,000	3,000	4,000	33
PRENGINE [6]	1,865	1,908	3,258	180.3

The reconfiguration time for including a new FU into the *Reconfigurable Part* depends on two previous concepts: the bitrate of the *prEngine* component and the characteristic of the module to be deployed. Thus, TABLE III. 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.

TABLE III. TABLE STYLES

COMPONENTS	TIME (µS)
1 FU component	249.15
2 FU components	487.47
3 FU components	731.20

VI. CONCLUSIONS

This paper presents a dynamic and partially reconfigurable embedded system for hyperspectral image processing that allows extracting the endmembers of a Cuprite image according to the MVCA algorithm, onto a Xilinx Virtex5-LX110T FPGA. The efficiency of the proposed system 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 180MB/s. In regards to the scalability, the system might modify its internal structure by deploying new FU components at run-time without re-synthezing the system. Even more, the application might be scaled by replicating the computation element (FU) into the reconfigurable region.

The proposed system is able to support real-time processing by considering the AVIRIS rates. It should be noted that its cross-track line scan time is 8.3 ms to collect 512 full pixel vectors. This introduces the need to process an AVIRIS Cuprite scene of 350×350 pixels and 224 bands in less than 1.985s to fully achieve real-time performance. In our case, the Cuprite scene of 250×191 pixels and 14 bands should be processed in less than 56.835ms. 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.

REFERENCES

[1] R. O. Green, M. L. Eastwood, C. M. Sarture, T. G. Chrien, M. Aronsson, B. J. Chippendale, J. A. Faust, B. E. Pavri, C. J. Chovit, M. Solis et al., "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer (AVIRIS)," Remote Sensing of Environment, vol. 65, no. 3, pp. 227–248, 1998.

[2] J.M.P. Nascimento, J.M.B. Dias; "Vertex component analysis: a fast algorithm to unmix hyperspectral data", IEEE Trans. on Geosc. and Remote Sensing, vol.43, no.4, pp. 898- 910, 2005.

[3] S. López, P. Horstrand, G.M. Callicó, J.F. López, and R. Sarmiento R.; "A low-computational-complexity algorithm for hyperspectral endmember extraction: Modified Vertex Component Analysis"; IEEE Geosc. and Remote Sensing Letters; vol.9, no.3, pp.502-506, 2012.

[4] S. Bernabe, S. Sanchez, A. Plaza, S. Lopez, J. A. Benediktsson and R. Sarmiento, "Hyperspectral Unmixing on GPUs and Multi-Core Processors: A Comparison," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 6, no. 3, pp. 1386-1398, 2013.

[5] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader and J. Chanussot, "Hyperspectral unmixing overview: geometrical, statistical and sparse regression-based approaches," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 5, no. 2, pp. 354-379, 2012.

[6] J. Dondo, J. Barba, F. Rincón, F. Moya, J.C. Lopez, "Dynamic Objects: Supporting Fast and Easy Run-Time Reconfiguration in FPGAs", Journal of Systems Architecture, 2013.

[7] E. Gamma, R. Helm, R. Johnson, J.M. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, pp.99-110, 1995.

[8] Xilinx, "LogiCORE IP XPS HWICAP Product Specification", Xilinx 2010.

[9] M. Hubner, D. Gohringer, J. Noguera, J. Becker, "Fast dynamic and partial reconfiguration Data Path with low Hardware overhead on Xilin FPGAs", IEEE Int. Symposium on Parallel and Distributed Processing, 2010.

[10] S. Lamonnier, M. Thoris, M. Ambielle, "Accellerate Partial Reconfiguration with a 100\% Hardware Solution", Xcell, 2012.