INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Mestrado em Engenharia de Electrónica e Telecomunicações (Perfil de Telecomunicações)

Implementação em hardware reconfigurável de método de separação de dados hiperespetrais

João Pedro José do Rosário

(Licenciado em Engenharia Electrónica e Telecomunicações e de Computadores)

Relatório do Trabalho Final de Mestrado para Obtenção do Grau de Mestre em Engenharia de Electrónica e Telecomunicações

Orientadores:

Professor Doutor Mário Pereira Véstias

Professor Doutor José Manuel Peixoto Nascimento

Júri:

Presidente: Professora Doutora Maria Manuela Almeida Carvalho Vieira

Vogal: Professor Doutor Mário Pereira Véstias

Professor Doutor Vítor Manuel Mendes Silva

Dezembro de 2014

Agradecimentos

Em primeiro lugar, quero agradecer aos meus orientadores, Professor Mário Véstias e Professor José Nascimento, pelo apoio e orientação que me deram ao longo deste trabalho.

Quero agradecer também à Fundação para a Ciência e a Tecnologia e ao Instituto de Telecomunicações, por ter disponibilizado o equipamento necessário para realizar este trabalho, sendo este suportado pelo projeto PEst-OE/EEI/LA0008/2013.

Ao Grupo de Investigação em Eletrónica e Sistemas de Telecomunicações, por me ter disponibilizado um local de trabalho no seu laboratório de investigação.

A todos os meus colegas e amigos que me acompanharam durante este trabalho, que partilharam e discutiram algumas ideias comigo e também a todos os momentos de convívio partilhados com eles.

Um agradecimento muito especial aos meus pais, irmã e família, pela sua compreensão durante esta fase da minha vida e também pelos seus incentivos nesta longa caminhada.

Por fim, agradeço à minha namorada Filipa, pelo apoio e carinho nesta fase da minha vida.

Resumo

Os sensores hiperespetrais adquirem grandes quantidades de dados com uma elevada resolução espetral. Esses dados são utilizados em aplicações para classificar uma área da superfície terrestre ou detetar um determinado alvo. No entanto, existem aplicações que requerem processamento em tempo-real. Recentemente, sistemas de processamento a bordo têm surgido para reduzir a quantidade de dados a ser transmitida para as estações base e assim reduzir o atraso entre a transmissão e a análise dos dados. Sistemas esses compactos, com *hardware* reconfigurável, como os *field programmable gate arrays* (FPGAs).

O presente trabalho propõe uma arquitetura num FPGA, que paraleliza o método *vertex components analysis* (VCA) de separação de dados hiperespetrais. Este trabalho é desenvolvido na placa ZedBoard que contém um Xilinx Zynq®-7000 XC7Z020.

Na primeira fase realiza-se uma análise ao desempenho do método sem o pre-processamento de redução de dados, em termos espetrais. O método é otimizado, para reduzir o seu peso e complexidade computacional. O processo de ortogonalização é a parte mais pesada do método, é realizada por uma decomposição de valores singulares (singular value decomposition - SVD). Este processo é simplificado por uma decomposição QR que reutiliza os vetores ortogonais já determinados. É ainda analisado o tipo de precisão que o método necessita para manter o mesmo desempenho e é concluído que necessita de pelo menos 48-bit vírgula fixa ou flutuante 32-bit.

Na segunda fase projeta-se uma arquitetura que paraleliza o método otimizado. Esta é escalável e consegue processar vários píxeis e/ou bandas espetrais em paralelo.

A arquitetura é implementada e dimensionada para o sensor AVIRIS, onde este captura 512 píxeis com 224 bandas espetrais em 8,3 ms e a arquitetura processa 614 píxeis e determina oito assinaturas espetrais em 1,57 ms, ou seja, a arquitetura implementada é apropriada para processamento em tempo-real de dados hiperespetrais.

Palavras-chave: Separação Espetral Linear, Determinação de *endmembers*, *Vertex Component Analysis* (VCA), *Field-Programmable Gate Array* (FPGA), Sistemas de Processamento a bordo.

Abstract

The Hyperspectral sensors acquire large datasets with high spectral resolution. These datasets are used to classify or detect a specific target over an area of Earth surface. However, there are applications that require real-time processing. Recently, on-board processing systems have emerged to reduce the amount of data that is transmitted to the ground base stations and thereby reduce the delay between the transmission and data analysis. On-board systems need to be compact, such as field programmable gate arrays (FPGAs).

This work presents a FPGA architecture, that parallels the *vertex components analysis* (VCA) method for hyperspectral unmixing data. This work is developed on a ZedBoard board, which contains a Xilinx Zynq®-7000 XC7Z020.

In the first phase an analysis of the method's performance without dimensionality reduction pre-processing step, in spectral terms, is conducted. The method have been also optimized, to reduce its computational weight and complexity. The orthogonal process, performed on the singular value decomposition (SVD) used in the original method, is the most complex part of the algorithm. This process is simplified using a QR decomposition that reuses the orthogonal vectors already determined. Its also analysed the type of precision that the method needs to maintain the same performance. In the present work it is concluded that the method requires at least 48-bit fixed-point or 32-bit floating-point.

In the second phase is projected an architecture that parallels the optimized method, which is scalable and can process multiple pixels and/or spectral bands in parallel.

The architecture is implemented and dimensioned to AVIRIS sensor, which acquires 512 pixels with 224 spectral bands in 8,3 ms, the architecture processes 614 pixels and extracts eight spectral signatures in 1,57 ms, therefore one can conclude that the implemented architecture is appropriated for real-time hyperspectral data processing.

Keywords: Linear Hyperspectral Unmixing, Endmember Extraction, Vertex Component Analysis (VCA), Field-Programmable Gate Arrays (FPGA), Onboard Processing.

Índice

Lista de Tabelas

Lista de Figuras

Lista de Acrónimos

ACC	Acumulador
ADD	Somador
ARM	Advanced RISC Machine
ED	Euclidean Distance
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
MUL	Multiplicador
MUX	Multiplexador
PC	Personal Computer
PPI	Pixel Purity Index
RMS	Root Mean Square
SAM	Spectral Angular Measure
SGA	Simplex Growing Algorithm
SID	Spectral Information Divergence
SNR	Signal-to-Noise Ratio
SoC	System on Chip
SUB	Subtrator
SVD	Singular Value Decomposition

USGS United States Geological Survey

VCA Vertex Component Analysis

Lista de Símbolos

Notação geral

a, A	letras minúsculas e maiúsculas designam escalares
$\mathbf{a}$	letras a negrito minúsculas designam vetores
$\mathbf{A}$	letras a negrito maiúsculas designam matrizes

Operadores

$[a_1, a_2, ..., a_L]$	vetor $1 \times L$ com os elementos a_i, $i = 1, ..., L$
$[\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_L]$	matriz com colunas a_i, $i = 1, ..., L$
$\mathbf{a}_i$	i-ésimo elemento do vetor $\mathbf{a}$
$arg\ max(\mathbf{a})$	argumento máximo do vetor $\mathbf{a}$
$[\mathbf{A}]_{:,j}$	j-ésima coluna da matriz $\mathbf{A}$
$\sum$	operador somatório
$\prod$	operador produtório
$\|.\|$	operador módulo
$(.)^T$	operador transposição
$E(.)$	operador esperança
$\mathbf{A}^+$	matriz pseudo-inversa de $\mathbf{A}$

Símbolos principais

α	abundância
p	número de *endmembers*
L	número de bandas espetrais numa imagem
N	número de pixeis numa imagem
$\mathbf{m}$	assinatura espetral
$\mathbf{e}$	vetor unitário
$\mathbf{f}$	direção ortogonal
$\mathbf{n}$	ruído aditivo
$\mathbf{v}$	valores da projeção
$\mathbf{w}$	vetor aleatório com distribuição gaussiana com média nula
$\mathbf{y}$	píxel
$\mathbf{A}$	*endmembers* determinados
$\mathbf{Q}$	base ortogonal da decomposição QR
$\mathbf{R}$	matriz triangular superior da decomposição QR
$\mathbf{W}$	ruído aditivo
$\mathbf{Y}$	imagem hiperespetral

1 Introdução

1.1 Enquadramento

Deteção remota é a ciência que obtém informação sobre uma determinada área da superfície terrestre, sem qualquer contacto físico com a mesma, através de instrumentos de deteção remota incorporados em plataformas aéreas (aeronaves) e espaciais (satélites). Esta informação depois de adquirida é processada e analisada para que as aplicações possam produzir resultados, como por exemplo na deteção de um determinado alvo ou na classificação da área em causa. Nos últimos anos esta ciência tem evoluído significativamente despertando grande interesse em diversas aplicações no campo civil [1]: na exploração de recursos naturais e mineral, controlo da agricultura e florestal, monitorização do ambiente e deteção/prevenção de desastres naturais ou industriais. No campo militar [2]: recolha de informação do campo de batalha; discriminação entre alvos reais ou falsos e deteção de alvos camuflados, tanques, mísseis e minas.

De entre os vários instrumentos de deteção remota destacam-se os sensores hiperespetrais que adquirem a luz solar refletida pela superfície terrestre, designada por radiância, e produzem várias imagens em simultâneo, como ilustra a Figura 1.1. Cada imagem adquirida tem duas dimensões espaciais e correspondem a um comprimento de onda diferente. Este conjunto de imagens forma uma estrutura de dados conhecida por cubo ou imagem hiperespetral, sendo cada píxel um vetor, cujo cada elemento representa a radiância, num dado comprimento de onda, de uma área da superficie observada, como ilustra a Figura 1.2.

Um sensor hiperespetral é caracterizado segundo os seguintes parâmetros:

- **Cobertura espetral** – região do espetro que o sensor adquire a radiância, desde do visível ao infravermelho, tipicamente entre 0,4 μm a 2,5 μm.

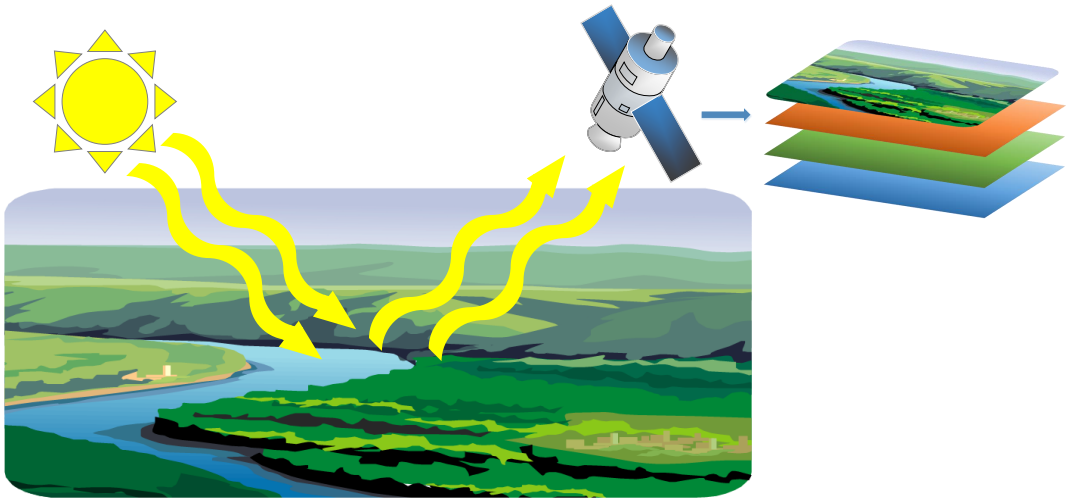


Figura 1.1: Processo de aquisição de imagens.

- **Resolução espetral** – capacidade de discriminar um comprimento de onda, na ordem dos 10 nm.
- **Resolução espacial** – dimensão espacial de um píxel, na ordem dos metros e dezenas de metros, sendo esta definida pela distância que o sensor se encontra da superficie terrestre e pelo seu campo de visão instantâneo.
- **Resolução radiométrica** – número de bits que a radiância é quantificada, entre 12 e 16 bits.
- **Número de bandas espetrais** – número de comprimentos de onda que o sensor discrimina, na ordem das dezenas e centenas.
- **Relação sinal-ruído** – relação entre a radiância medida e o ruído introduzido pela eletrónica do sensor.

A Tabela 1.1 apresenta o número de bandas espetrais, a cobertura espetral, a resolução espetral e espacial de oito sensores: quatro incorporados em aeronaves (AVIRIS [3], HYDICE [4], HyMap [5] e APEX) [6] e quatro em satélites (HYPERION [7], HySI [8], HICO [9] e HyspIRI [10]). Atualmente estão a ser realizados testes com o sensor HyspIRI numa aeronave e só no futuro estará operacional num satélite [11].

Estes sensores fazem a aquisição das imagens hiperespectrais segundo duas técnicas conhecidas por *whisk broom* ou *push broom*. A técnica *whisk broom* é usada pelos sensores AVIRIS e HyMap e faz a aquisição através de um dispositivo mecânico

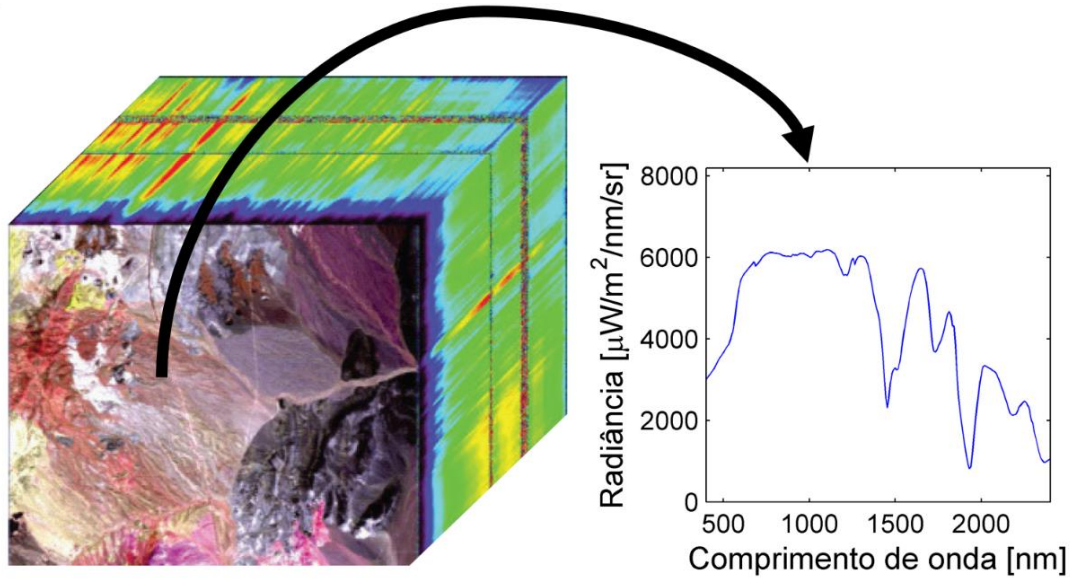


Figura 1.2: Representação do cubo hiperespetral e a radiância de um píxel.

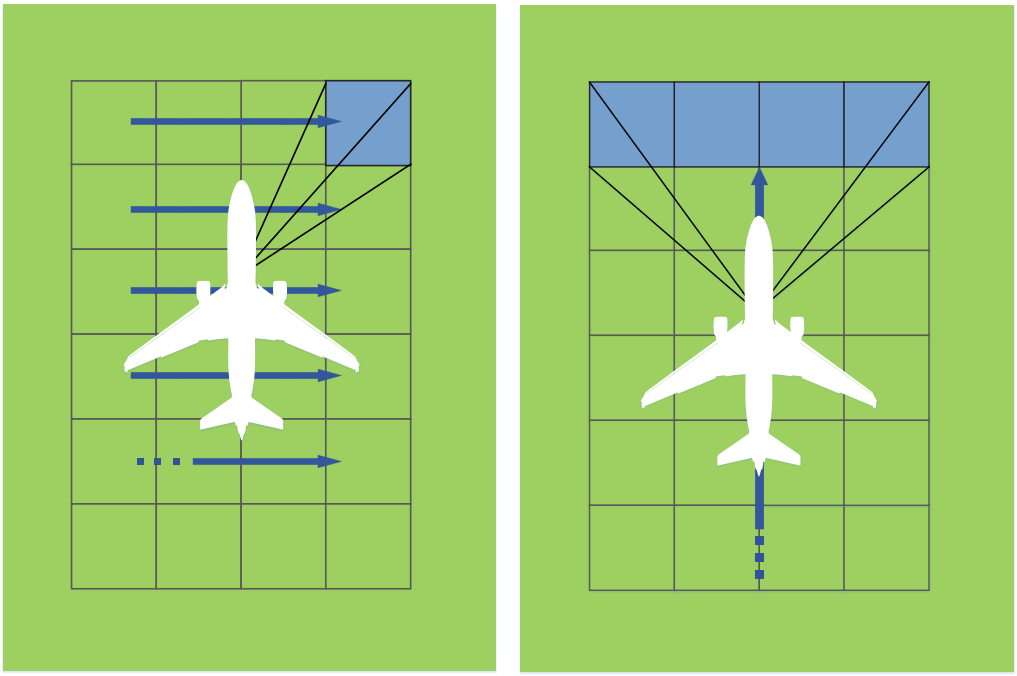
Sensor	**Número de bandas espetrais**	**Cobertura espetral** $[\mu m]$	**Resolução espetral** $[nm]$	**Resolução espacial** $[m]$
AVIRIS	224	0,4-2,5	10	20
HYDICE	210	0,4-2,5	10	0,75
HyMap	128	0,4-2,5	15	2-10
APEX	532	0,4-2,5	10	2-5
HYPERION	220	0,4-2,5	10	30
HySI	64	0,4-1,0	10	80
HICO	128	0,4-1,0	5,7	90
HyspIRI	217	0,4-2,5	4-12	60

Tabela 1.1: Características espetrais de oito sensores hiperespectrais.

oscilatório que captura um píxel de cada vez, como ilustra a Figura 1.3a. A técnica *push broom* é usada pelos sensores APEX, HYDICE, HYPERION, HySI, HICO, HyspIRI e faz a aquisição através de um dispositivo eletrónico que captura uma linha de píxeis ou várias linhas em simultâneo, como ilustra a Figura 1.3b.

1.2 Motivação

Os sensores hiperespetrais produzem grandes quantidades de informação num curto espaço de tempo. Por exemplo o sensor HYPERION produz uma imagem de $256 \times 6925 \times 242$ em 30 segundos [12], onde cada píxel é quantificado com 12 bits, o que resulta numa imagem com mais de 600 MB, e estima-se que o sensor HyspIRI



(a) Whiskbroom (b) Pushbroom

Figura 1.3: Técnicas de aquisição de dados: (a) *whisk broom*; (b) *push broom*.

produza diariamente cerca de 4,5 TB [13], ou seja, cerca de 1,6 GB em 30 segundos. No entanto, o uso desta informação põe em causa as aplicações que requerem processamento em tempo-real. Devido à quantidade de informação que tem de ser transmitida para a estação base, responsável por essa análise, e ao débito de *downlink* que é limitado e tem-se mantido constante nos últimos anos [13]. Por esta razão é necessário arranjar soluções que permitam reduzir a quantidade de informação a transmitir.

Uma solução possível passa por agregar a estas plataformas sistemas de alto desempenho computacional. Sistemas esses que processem uma ou várias técnicas de análise em tempo-real, de modo a ser apenas transmitida, à estação base, a informação necessária para determinada aplicação. Estes sistemas de alto desempenho são: *graphics processing units* (GPUs) [14] e *field-programmable gate arrays* (FPGAs) [15–17]. Estes necessitam de ser compactos, leves, tenham um baixo consumo energético e no caso das plataformas espaciais imunes à radiação espacial. Contudo, é desejável ainda que estes sistemas sejam flexíveis, para as plataformas se adaptarem às necessidades de diferentes aplicações. De entre os vários sistemas, os FPGAs são os que melhor se enquadram nestas características [18]. Nos últimos

anos, a comunidade científica tem proposto diversas implementações em FPGA de técnicas de análise a estas imagens, nomeadamente na separação de dados hiperespetral [19–26], deteção de alvos [27], *clustering* [28], compressão de imagens [29], entre outros.

1.3 Objetivos e contribuições

Esta dissertação propõe a elaboração de uma arquitetura que paralelize o algoritmo de um método de separação de dados hiperespetrais num FPGA, de modo a que no futuro possa ser integrada numa plataforma e permitindo assim processar a informação adquirida em tempo-real.

Para atingir o objetivo desta dissertação esta contemplou duas fases na sua realização. A primeira fase consistiu em analisar o comportamento do algoritmo, com recurso a imagens sintéticas, em função dos seguintes parâmetros: número de píxeis da imagem, relação sinal-ruído (*signal-to-noise ratio* - SNR) e número de substâncias a determinar. Esta também consistiu em analisar as propostas com o objetivo de otimizar o método e reduzir a sua complexidade computacional. A segunda fase consistiu em projetar e implementar um arquitetura que paralelize o método, com base nas propostas analisadas, e por fim avaliar o seu desempenho.

Este trabalho é desenvolvido na placa de desenvolvimento ZedBoard [30] que tem um Xilinx Zynq®-7000 XC7Z020, para testar e avaliar o desempenho da arquitetura implementada. Parte deste trabalho desenvolvido é publicado na conferência SPIE Remote Sensing 2014 em [20] e presente no apêndice A.

1.4 Organização do documento

Este documento está organizado em seis capítulos.

O segundo capítulo apresenta os fundamentos teóricos abordados na dissertação e o método de estado da arte que se pretende implementar.

O terceiro capítulo apresenta a análise do desempenho do método e da sua otimização.

O quarto capítulo apresenta a arquitetura proposta para paralelizar o algoritmo, juntamente com todos módulos desenvolvidos.

O quinto capítulo apresenta a os resultados da implementação da arquitetura implementada (recursos utilizados e tempo de execução) e a avaliação dos resultados obtidos do método na arquitetura.

O sexto capítulo sumariza o conteúdo apresentado em todos os capítulos, bem como as suas principais conclusões, e também apresenta um conjunto de sugestões à arquitetura para trabalho futuro.

2 | Análise de imagens hiperespetrais

Uma imagem hiperespetral é composta por três dimensões, duas espaciais e uma espetral. Cada píxel representa um vetor de radiâncias de vários de comprimentos de onda, com uma resolução espetral muito grande, e representa uma área da superficie terrestre, cuja as dimensões dependem das características e altitude do sensor. Tipicamente, estas imagens têm uma baixa resolução espacial e cada píxel é normalmente uma mistura de várias substâncias, designadas por *endmembers*, ou seja, um píxel é constituído por uma fração ou percentagem de cada *endmember*, designada por abundância. Quando um píxel apenas representa uma substância este designa-se por píxel puro, caso contrário este designa-se por não puro. A Figura 2.1 ilustra uma imagem hiperespetral com a representação espetral de três píxeis, um píxel puro (água) e dois não puros (solo mais rochas e vegetação mais solo).

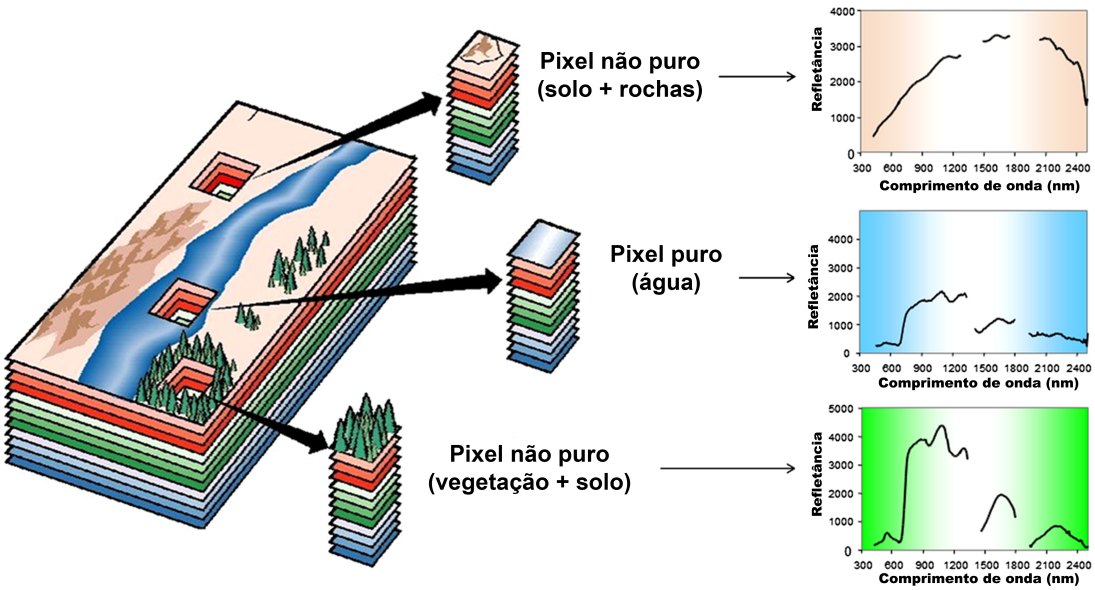


Figura 2.1: Imagem hiperespetral com a representação espetral de três píxeis. [31]

2.1 Modelos de mistura

As imagens hiperespetrais podem ser analisadas com base em dois modelos de mistura: linear ou não linear. O modelo linear [32] considera cada píxel uma combinação linear entre as assinaturas dos *endmembers* presentes nesse píxel, pesadas pelas suas abundâncias. Este modelo ignora a influência que pode existir entre os diferentes *endmembers*, como ilustra a Figura 2.2. O modelo não linear [33] considera cada píxel uma combinação não linear entre os vários *endmembers* presentes e tem em consideração a influência que pode existir entre os diferentes *endmembers* desse píxel, como ilustra a Figura 2.3. Embora o modelo não linear seja mais realista, o modelo linear é frequentemente adotado, pois é mais simples e consegue produzir resultados com boa qualidade [34, 35].

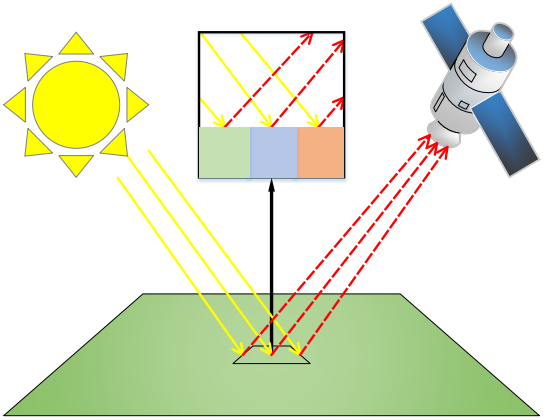


Figura 2.2: Modelo de mistura linear.

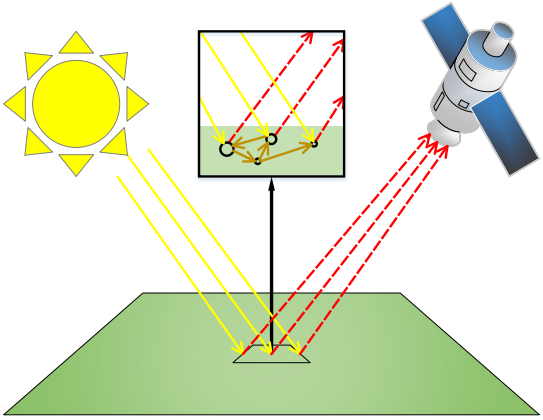


Figura 2.3: Modelo de mistura não-linear.

O modelo de mistura linear descreve um píxel, designado por um vetor $\mathbf{y}$ com dimensão L, onde L é o número de bandas espetrais, segundo a seguinte expressão:

$$\mathbf{y} = \sum_{i=1}^{p} \alpha_i \mathbf{m}_i + \mathbf{n}, \tag{2.1}$$

onde $\mathbf{m}_i$ representa a assinatura espetral do i-ésimo *endmember*, α_i representa a abundância do i-ésimo *endmember*, p representa o número de *endmembers* e $\mathbf{n}$ representa o ruído aditivo que pode existir. A Figura 2.4 ilustra um exemplo do modelo, sendo considerado um espaço a três dimensões e os píxeis da imagem formam uma estrutura de dados designada por *simplex*. Este *simplex* é definido pelo píxeis puros $\mathbf{m}_1$, $\mathbf{m}_2$ e $\mathbf{m}_3$ que se encontram nos vértices, e os restantes píxeis não puros encontram-se dentro do *simplex*.

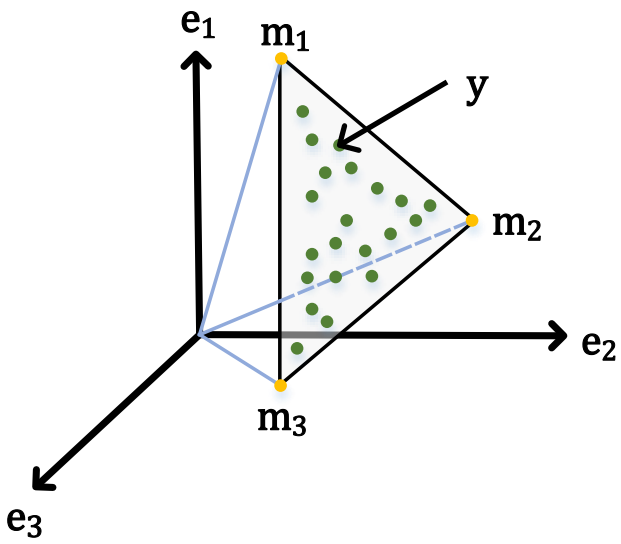


Figura 2.4: Exemplo de um *simplex* formado pelos píxeis de uma imagem segundo o modelo linear de mistura.

2.2 Separação hiperespetral

A separação hiperespetral é uma técnica de análise de imagens hiperespetrais. Esta consiste na decomposição do espetro de uma imagem num conjunto de assinaturas espetrais dos *endmembers* que a constituem e a respetiva abundância de cada

endmember. A Figura 2.5 apresenta a constituição da técnica de separação hiperespetral:

- **Correção atmosférica** – converte a imagem de radiâncias para refletâncias, aplicando um modelo de correção atmosférico [36, 37], de modo a compensar os efeitos atmosféricos (dispersão e absorção) que podem influenciar a imagem adquirida. [38]

- **Redução de dados** – reduz a dimensão espetral da imagem, através da projeção dos dados num determinado subespaço de menor dimensão. Este passo aumenta o desempenho computacional dos próximos passos, reduz também o espaço necessário para guardar a imagem e melhora a qualidade dos resultados, devido ao aumento da SNR na imagem, pois o número de *endmembers* presentes na imagem é muito menor do que o número de bandas espetrais que a imagem tem [39].

- **Separação espetral** – é constituída por dois passos: determinação das assinaturas espetrais de cada *endmember* e o segundo estima o mapa de abundâncias de cada *endmember* determinado.

Contudo, a correção atmosférica e a redução de dados podem ser opcionais para alguns métodos de análise hiperespetral [40].

Na separação hiperespetral existem métodos geométricos que determinam os *endmembers* presentes na imagem. Estes consideram a existência de pelo menos um píxel puro por cada *endmember*, ou seja, existe pelo menos um vetor espetral em cada vértice do *simplex*. Os métodos tiram ainda partido de uma das seguintes propriedades das assinaturas de cada *endmember*:

1 – os extremos de uma projeção de dados em qualquer direção corresponde aos *endmembers*,

2 – o volume definido por qualquer conjunto de $\mathbf{p}$ vetores espetrais é máximo quando estes vetores correspondem os *endmembers*.

Os métodos que tiram partido da primeira propriedade são:

- *pixel purity index* (PPI) [41] implementado num FPGA em [21] e [22, 23],

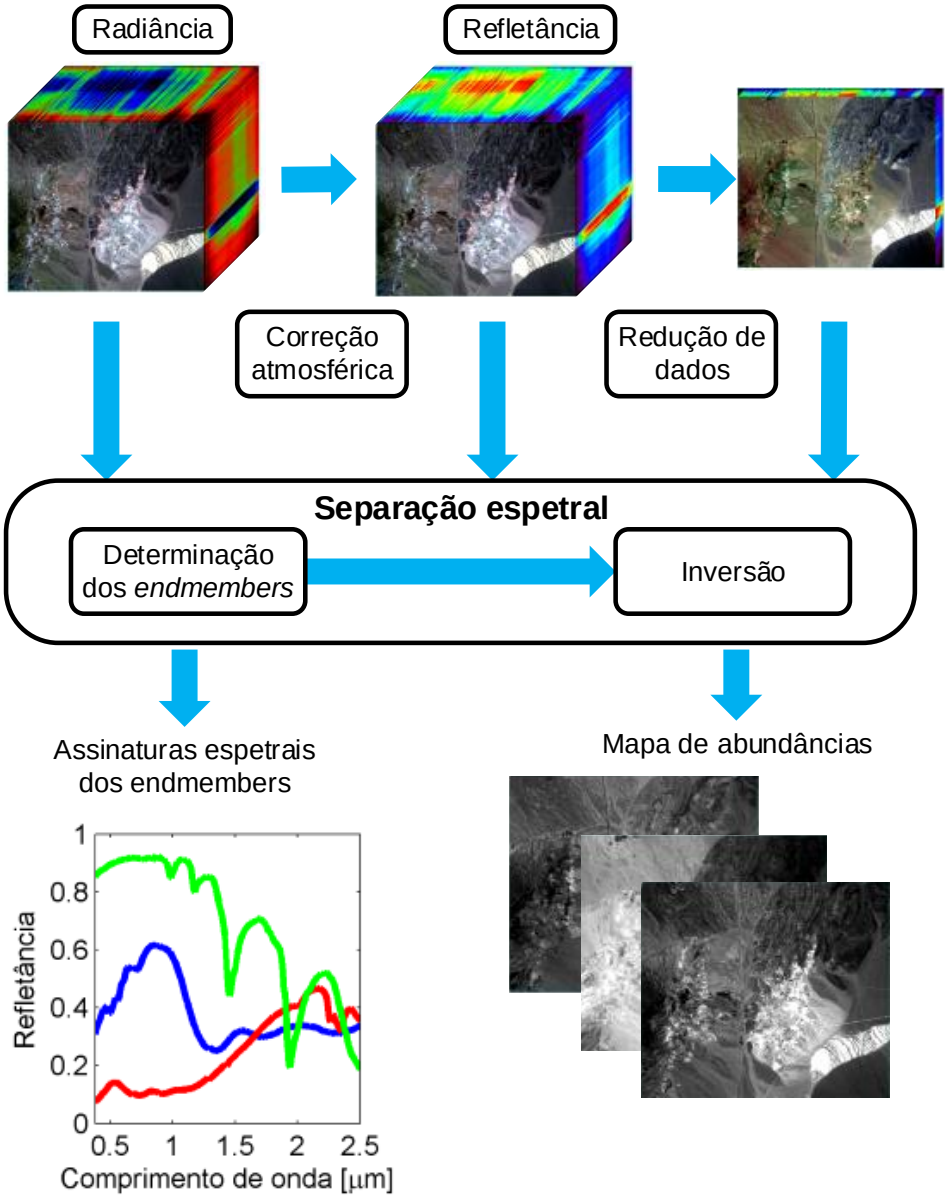


Figura 2.5: Processo da técnica de análise de separação hiperespetral.

- *vertex component analysis* (VCA) [40] implementado num FPGA em [19] e [20], ambos com modificações para otimizar o seu algoritmo. No entanto em [19] considera-se o pre-processamento de redução de dados e em [20] não se considera.

Os métodos que tiram partido da segunda propriedade são:

- N-FINDR [42] implementado num FPGA em [24],
- *simplex growing algorithm* (SGA) [43] implementado num FPGA em [25] com

modificações para otimizar o seu algoritmo.

2.3 Método de determinação de *endmembers*

O VCA é um método de estado da arte de separação de dados hiperespetral, que determina os *endmembers* presentes na imagem. Este é automático, iterativo e funciona com ou sem a correção atmosférica e a redução de dados da análise hiperespetral. O método consiste em iterativamente projetar os dados numa direção ortogonal ao subespaço gerado pelos *endmembers* já determinados. O novo *endmember* corresponde ao píxel com a máxima amplitude absoluta dessa projeção. Este processo repete-se até serem determinados todos os *endmembers*. A Figura 2.6 ilustra o funcionamento do método num *simplex* de definido por três *endmembers*. Na primeira iteração os dados são projetado na primeira direção $\mathbf{f}_1$, onde o extremo dessa projeção corresponde ao *endmember* $\mathbf{m}_a$. Na iteração seguinte, o *endmember* $\mathbf{m}_b$ é determinado pela projeção da direção $\mathbf{f}_2$, ortogonal a $\mathbf{m}_a$. Por fim, uma nova direção $\mathbf{f}_3$ é gerada, ortogonal ao subespaço gerado pelos *endmembers* já determinados e o *endmember* $\mathbf{m}_c$ é determinado através do extremo da projeção em $\mathbf{f}_3$.

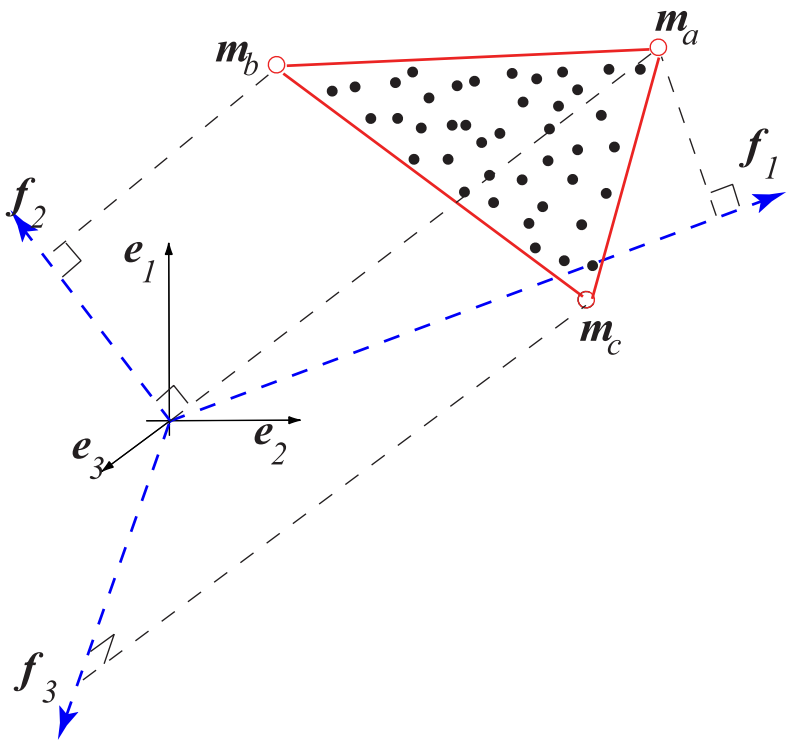


Figura 2.6: Processo de determinação de três *endmembers* do algoritmo VCA.

O Algoritmo 1 apresenta o método VCA, sem o pré-processamento de redução de dados. O algoritmo recebe como parâmetros de entrada uma matriz $\mathbf{Y}$ de dimensão $L \times N$, onde L o número de bandas espetrais e N o número de píxeis. Esta matriz reorganiza a imagem hiperespetral, onde as colunas são os píxeis e as linhas as bandas espetrais da imagem. O método também recebe o número de *endmembers* a determinar, representado por p. Este começa por iniciar a matriz $\mathbf{A}$ de dimensão $L \times p$, responsável por armazenar os *endmembers* determinados, com um vetor unitário $\mathbf{e_u}$. A cada iteração do algoritmo é gerado um vetor aleatório, representado pelo vetor $\mathbf{w}$ de dimensão L, com distribuição gaussiana e média nula. Depois é gerado um vetor ortogonal à matriz $\mathbf{A}$, representado por $\mathbf{f}$ com dimensão L, através da expressão apresentada na linha 5, onde $\mathbf{I}$ é uma matriz identidade e $\mathbf{A}^+$ a matriz pseudo-inversa da matriz $\mathbf{A}$ ambas de dimensão $L \times L$. De seguida são projetados os dados da imagem $\mathbf{Y}$ no vetor $\mathbf{f}$, sendo o resultado armazenado no vetor $\mathbf{v}$ de dimensão N. Depois é determinado o argumento máximo dessa projeção, representado por k, e é armazenado na coluna i da matriz $\mathbf{A}$ a coluna k da matriz $\mathbf{Y}$, sendo este o vetor correspondente ao novo *endmember* determinado. Este processo iterativo repete-se p vezes e retorna a matriz $\mathbf{A}$ com os *endmembers* determinados.

Algorithm 1 VCA

1: **ENTRADA** $\mathbf{Y}$, p
2: $\mathbf{A} := [\mathbf{e_u}|\mathbf{0}|...|\mathbf{0}]; \{\mathbf{e_u} := [0, ..., 0, 1]^T\}$
3: **for** $i := 1$ to p **do**
4: $\quad \mathbf{w} := \mathrm{rand}(L, 1)$;
5: $\quad \mathbf{f} := (\mathbf{I} - \mathbf{A}\mathbf{A}^+)\mathbf{w}$;
6: $\quad \mathbf{v} := \mathbf{f}^T\mathbf{Y}$;
7: $\quad k := \arg\max_{j=1,...,N} |[\mathbf{v}]\ :, j|$;
8: $\quad [\mathbf{A}]_{:,i} := [\mathbf{Y}]_{:,k}$;
9: **end for**
10: **SAÍDA** $\mathbf{A}$

O processo de cálculo do vetor $\mathbf{f}$ é computacionalmente exigente, pois a cada iteração do método é necessário calcular a matriz pseudo-inversa $\mathbf{A}^+$ da matriz $\mathbf{A}$. Tipicamente, essa matriz é determinada através da decomposição em valores singulares (*singular value decomposition* - SVD) [44], como é demostrado no trabalho relativo ao VCA [40]. Este processo é simplificado por uma decomposição QR com base no processo de Gram-Schmidt [44–46]. No capítulo seguinte apresenta-se a simplificação proposta ao método VCA.

3 | Análise do método de determinação de *endmembers*

A análise ao método VCA verifica o seu desempenho, com base nos seguintes parâmetros: número de píxeis da imagem, SNR da imagem e número de *endmembers* a determinar. O método é otimizado com o objetivo de reduzir o seu peso e complexidade computacional e analisa-se o tipo de precisão (vírgula flutuante ou fixa) necessário para este manter o mesmo desempenho que o método original. Esta análise tem por base os seguintes parâmetros: SNR da imagem e número de *endmembers*. Em cada análise realiza-se um conjunto de experiências, com recurso a imagens sintéticas, onde a assinatura espetral de cada *endmember* é conhecida.

Em cada experiência são selecionadas, aleatoriamente, um determinado número de assinaturas espetrais. Estas estão numa base de dados digital disponibilizada pela instituição *United States Geological Survey* (USGS) [47]. Esta base de dados tem as refletâncias de vários minerais e as suas assinaturas têm 224 bandas espetrais ($L = 224$). As abundâncias dos dados da imagem geram-se a partir de uma distribuição de Dirichlet [48] dada pela seguinte função densidade de probabilidade:

$$p(\alpha_1, \alpha_2, ..., \alpha_p) = \frac{\Gamma(\sum_{i=1}^{p} \mu_i)}{\prod_{i=1}^{p} \Gamma(\mu_i)} \prod_{i=1}^{p} \alpha_i^{\mu_i - 1}, \tag{3.1}$$

onde $\alpha_i \in [0, 1]$, $\sum_{i=1}^{p} \alpha_i = 1$ e representa a abundância da *i*-ésima assinatura num píxel, $\Gamma(.)$ representa a função Gamma e μ_i é o parâmetro de concentração da *i*-ésima assinatura. Os dados gerados são representados por uma matriz $\mathbf{X}$ com dimensão $L \times N$, onde cada coluna é um vetor que representa um píxel sem ruído e segue o modelo de mistura linear:

$$\mathbf{x} = \sum_{i=1}^{p} \alpha_i m_i. \tag{3.2}$$

Aos dados é lhes adicionado ruído com uma determinada SNR dada por:

$$\mathrm{SNR}_{\mathrm{dB}} = 10 \log_{10} \left(\frac{\mathrm{E}(\mathbf{X}^T \mathbf{X})}{\mathrm{E}(\mathbf{W}^T \mathbf{W})} \right), \tag{3.3}$$

onde $\mathbf{W}$ é uma matriz com dimensão $L \times N$, cujo os valores são aleatórios com uma distribuição Normal com média nula e variância:

$$\sigma^2 = \frac{\mathrm{E}(\mathbf{X}^T \mathbf{X})}{L \times N \times 10^{SNR_{dB}/10}}. \tag{3.4}$$

Estas experiências não são comparáveis com outros métodos que recorrem ao pre-processamento de redução de dados espetrais, pois neste trabalho opta-se por não o considerar. Este pre-processamento muitas vezes requer determinar a matriz com os vetores próprios da matriz de correlação dos dados, sendo este processamento por vezes demorado, e pode por em causa o uso destes dados em aplicações que necessitam de processamento em tempo-real.

Determinados os *endmembers*, cada um é associado a uma assinatura selecionada. Esta associação é realizada pelo método Húngaro [49, 50].

Realizada a associação, cada par de assinaturas, representado pelos vetores $\mathbf{m}_a$ e $\mathbf{m}_b$ de dimensão L, é avaliada a semelhança entre elas com base em métricas de avaliação: medida angular espetral (*spectral angular measure* - SAM) [51–53], divergência da informação espetral (*spectral information divergence* - SID) [54] e a distância euclidiana (*euclidian distance* - ED) [52].

A SAM mede o ângulo formado entre as duas assinaturas e é definida pela seguinte expressão:

$$\mathrm{SAM}(\mathbf{m}_a, \mathbf{m}_b) = \arccos \left(\frac{\langle \mathbf{m}_a, \mathbf{m}_b \rangle}{||\mathbf{m}_a|| ||\mathbf{m}_b||} \right), \tag{3.5}$$

onde $\langle \mathbf{m}_a, \mathbf{m}_b \rangle = \mathbf{m}_a^T \mathbf{m}_b$, $||\mathbf{m}_a|| = \sqrt{\mathbf{m}_a^T \mathbf{m}_a}$ e $||\mathbf{m}_b|| = \sqrt{\mathbf{m}_b^T \mathbf{m}_b}$.

O SID mede a divergência entre duas assinaturas e é definida pela seguinte expressão:

$$\mathrm{SID}(\mathbf{m}_a, \mathbf{m}_b) = D(\mathbf{m}_a || \mathbf{m}_b) + D(\mathbf{m}_b || \mathbf{m}_a), \tag{3.6}$$

onde $D(\mathbf{m}_a || \mathbf{m}_b)$ e $D(\mathbf{m}_b || \mathbf{m}_a)$ representam a entropia relativa entre cada assinatura e

definidas por:

$$D(\mathbf{m}_a||\mathbf{m}_b) = \sum_{i=1}^{L} p_i \log\left(\frac{p_i}{q_i}\right) \tag{3.7}$$

e

$$D(\mathbf{m}_b||\mathbf{m}_a) = \sum_{i=1}^{L} q_i \log\left(\frac{q_i}{p_i}\right), \tag{3.8}$$

onde p_i e q_i são os i-ésimos elementos dos vetores $\mathbf{p}$ e $\mathbf{q}$ respetivamente. Os vetores $\mathbf{p}$ e $\mathbf{q}$ são designados por:

$$\mathbf{p} = \frac{\mathbf{m}_a}{\mathbf{m}_a^T \mathbf{1}} \tag{3.9}$$

e

$$\mathbf{q} = \frac{\mathbf{m}_b}{\mathbf{m}_b^T \mathbf{1}}, \tag{3.10}$$

onde $\mathbf{1}$ é um vetor cujo todos os elementos são unitários.

A ED mede a distância entre duas assinaturas e é definida pela seguinte expressão:

$$\mathrm{ED}(\mathbf{m}_a, \mathbf{m}_b) = 2\sqrt{1 - \cos(\mathrm{SAM}(\mathbf{m}_a, \mathbf{m}_b))}. \tag{3.11}$$

As métricas SAM e SID têm uma vantagem face à ED, pois estas ignoram a iluminação do píxel [54, 55].

No resultado de cada análise é apresentada a média de cada métrica de avaliação entre todos os *endmembers* determinados e a média da variância de cada conjunto de *endmembers*.

3.1 Desempenho do método

A análise do desempenho do método é realizada em função de três parâmetros: número de píxeis da imagem, SNR da imagem e número de *endmembers* a determinar. Nesta análise o método opera em vírgula flutuante com precisão dupla.

3.1.1 Análise em função do número de píxeis

Nesta análise são realizadas 100 experiências por cada imagem com número de píxeis diferente. Estas imagens têm 100, 1000 e 10000 píxeis, onde cada píxel é uma mistura de três assinaturas e as suas abundâncias são geradas com uma distribuição Dirichlet com os seguintes parâmetros $\mu_{1..3} = 1/3$ e uma SNR de 50 dB.

A Figura 3.1 apresenta a média e a variância da avaliação da SAM e a Tabela 3.1 apresenta a média e a variância da avaliação da SID e ED em função do número de píxeis. Os resultados demonstram um melhoramento no desempenho do método face ao aumento do número de píxeis processados, pois a probabilidade de existir píxeis puros também aumenta.

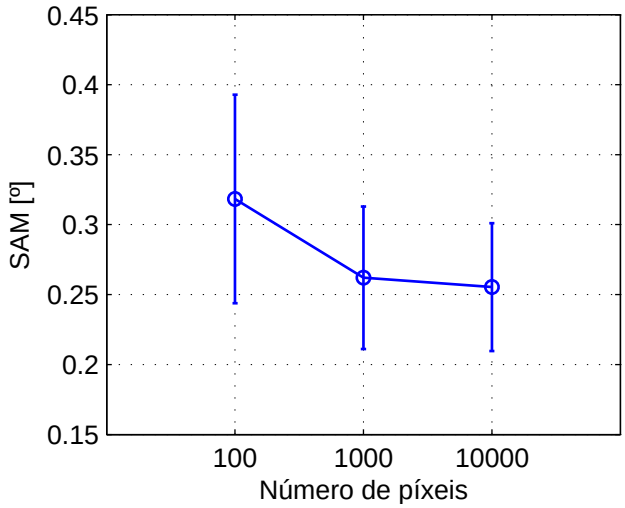


Figura 3.1: Média e variância dos resultados da avaliação da SAM das experiências em função do número de píxeis da imagem.

Métrica	$\mathrm{SID}/10^{-5}$	$\mathrm{ED}/10^{-3}$
100 píxeis	7,61(2,33e-03)	7,90(4,54e-02)
1000 píxeis	5,27(1,20e-03)	6,50(3,10e-02)
10000 píxeis	4,92(1,28e-03)	6,30(2,78e-02)

Tabela 3.1: Média e variância dos resultados da avaliação da SID e ED das experiências em função do número de píxeis da imagem.

Por vezes o método tem dificuldades em determinar o píxel puro de assinaturas com baixa energia, pois o ruído pode influenciar essa determinação. A Figura 3.2 apresenta as assinaturas espetrais utilizadas numa dessas experiências e a Figura 3.3 apresenta o *simplex* projetado no plano, onde o eixo das abcissas e das coordenadas representam a banda 110 e 190 respetivamente, os pontos a preto representam os píxeis, a linha azul representa o *simplex* correto formado pelos dados, com os píxeis puros assinalados com uma cruz, e a vermelho o *simplex* determinado, com os *endmembers* determinados assinalados com um círculo. Neste caso em particular, a SAM entre a assinatura 2 e a respetiva assinatura determinada é cerca de 1,7º.

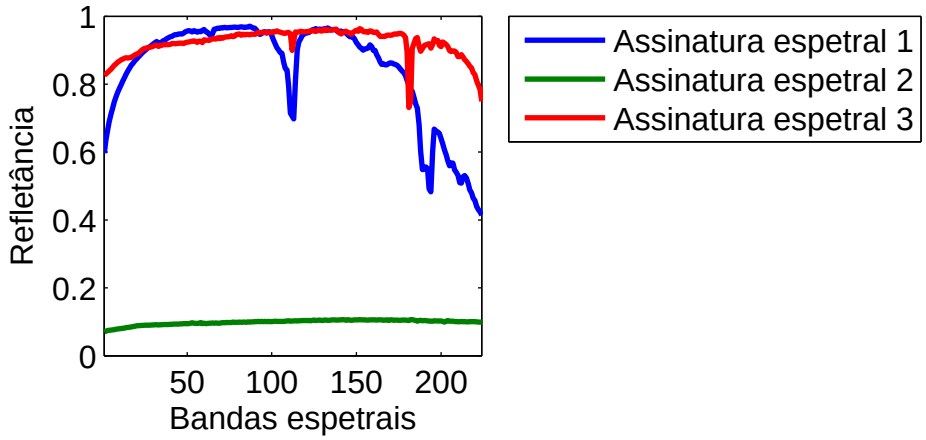


Figura 3.2: Assinaturas espetrais das três substâncias utilizadas numa experiência onde o método teve dificuldades em determinar um píxel puro como *endmember*.

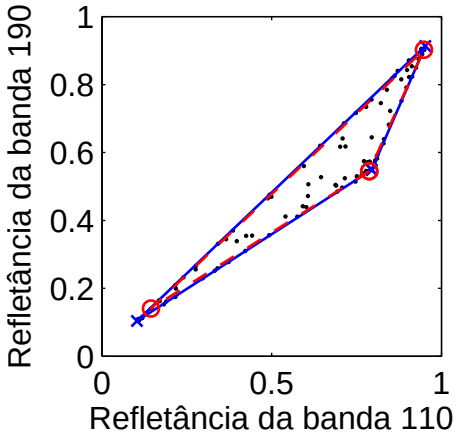


Figura 3.3: *Simplex* projetado no plano formado pelas bandas 110 e 190 da experiência onde o método teve dificuldades em determinar um píxel puro com baixa energia.

3.1.2 Análise em função da SNR

Nesta análise são realizadas 100 experiências por cada imagem com SNR diferente. Estas imagens têm uma SNR com 20, 30, 40, 50 e infinito dB, 10000 píxeis, onde cada píxel é uma mistura de três assinaturas e as suas abundâncias são geradas por uma distribuição Dirichlet com os seguintes parâmetros $\mu_{1..3} = 1/3$.

A Figura 3.4 apresenta a média e a variância da avaliação da SAM e a Tabela 3.2 apresenta a média e a variância da avaliação da SID e ED em função da SNR dos dados. Os resultados demonstram que o desempenho do método aumenta face ao aumento da SNR, pois a potência do ruído é menor.

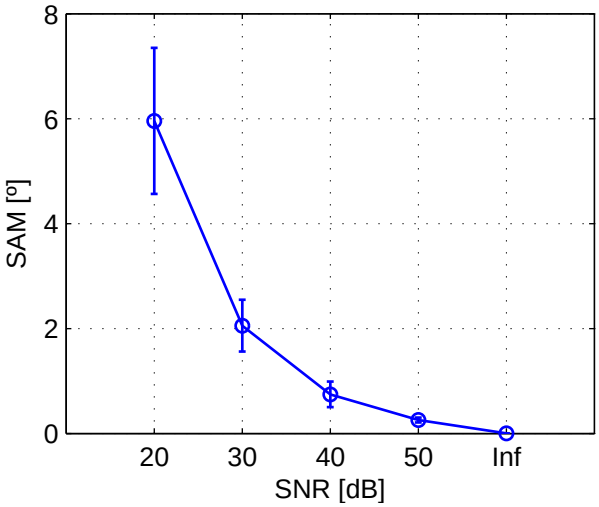


Figura 3.4: Média e variância dos resultados da avaliação da SAM das experiências em função da SNR dos dados.

Métrica	$\mathrm{SID}/10^{-4}$	$\mathrm{ED}/10^{-3}$
SNR 20 dB	1333,68(3,95e-01)	146,95(8,45e-01)
SNR 30 dB	185,61(3,00e-02)	50,71(3,01e-01)
SNR 50 dB	34,30(3,79e-03)	18,37(1,48e-01)
SNR 50 dB	4,92(1,28e-04)	6,30(2,78e-02)
SNR $+\infty$ dB	$\simeq$0(0)	$\simeq$0(0)

Tabela 3.2: Média e variância dos resultados da avaliação da SID e ED das experiências em função da SNR dos dados.

3.1.3 Análise em função do número de *endmembers*

Nesta análise são realizadas 100 experiências por cada imagem com número de assinaturas diferente. Estas imagens têm 10000 píxeis, onde cada píxel é uma mistura de p assinaturas ($p = 3, 6, 9, 12, 15, 18$) e as suas abundâncias são geradas po uma distribuição Dirichlet com os seguintes parâmetros $\mu_{1..p} = 0{,}1/p$ e uma SNR de 50 dB.

A Figura 3.5 apresenta a média e a variância da avaliação da SAM e a Tabela 3.3 apresenta a média e a variância da avaliação da SID e ED em função do número de *endmembers* a determinar. Os resultados demonstram que o desempenho do método mantém-se constante face ao aumento do número de assinaturas a determinar.

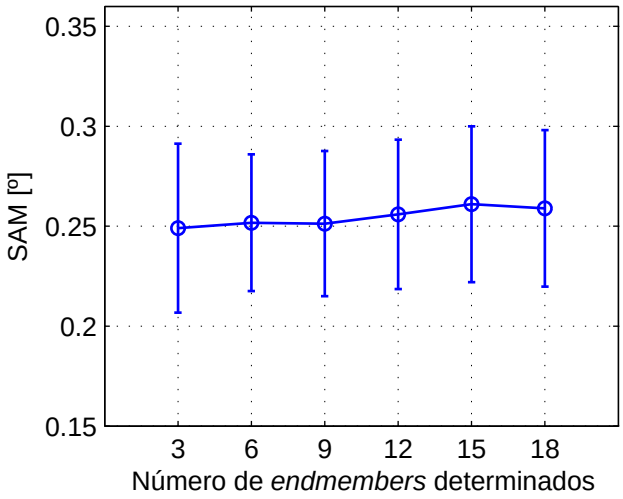


Figura 3.5: Média e variância dos resultados da avaliação da SAM das experiências em função do número de *endmembers* a determinar.

Métrica	$\mathrm{SID}/10^{-5}$	$\mathrm{ED}/10^{-3}$
3 *endembers*	4,32(8,81e-04)	6,10(2,58e-02)
6 *endembers*	4,39(7,30e-04)	6,21(2,08e-02)
9 *endembers*	4,84(1,07e-03)	6,22(2,21e-02)
12 *endembers*	4,87(8,97e-04)	6,31(2,28e-02)
15 *endembers*	5,37(1,10e-03)	6,44(2,38e-02)
18 *endembers*	5,05(9,85e-04)	6,39(2,38e-02)

Tabela 3.3: Média e variância dos resultados da avaliação da SID e ED das experiências em função da SNR dos dados.

3.2 Desempenho do método otimizado

O método VCA é otimizado para reduzir o seu peso e complexidade computacional. A tarefa mais complexa do método é o cálculo do vetor $\mathbf{f}$, pois esta requer calcular uma matriz pseudo-inversa. Esta é calculada através da SVD, que é numericamente bastante estável e precisa [56], mas é computacionalmente complexa e pesada. O processo de cálculo desse vetor é substituído por uma decomposição QR com base no processo de Gram-Schimdt clássico, sendo este menos estável que a SVD [56]. O algoritmo utilizado nesta decomposição é incremental, ou seja, este reutiliza os vetores ortogonais $\mathbf{Q}$ anteriormente calculados e a cada iteração apenas calcula o novo vetor ortogonal a incrementar na matriz $\mathbf{Q}$. O Algoritmo 2 apresenta o pseudo-código do método VCA otimizado. A análise do desempenho ao método otimizado é realizada em função de dois parâmetros: SNR da imagem e número de *endmembers* a

Algorithm 2 VCA otmizado

ENTRADA $\mathbf{Y}, p$
$\mathbf{w} := rand(L, 1)$;
$\mathbf{f} := \mathbf{w}$;
$j := 1$;
for $i := 1$ to p **do**
  if $i > 1$ **then**
    $\mathbf{temp} := [\mathbf{Y}]_{:,\mathbf{idx}_{(i-1)}}$;
    if $i > 2$ **then**
      $j := j + 1$;
      $\mathbf{r} := [\mathbf{Q}]^T_{:,1..(j-1)}[\mathbf{Y}]_{:,\mathbf{idx}_{(i-1)}}$;
      $\mathbf{temp} := \mathbf{temp} - \mathbf{r}[\mathbf{Q}]_{:,1..(j-1)}$;
    end if
    $r_{last} := 1/\sqrt{\mathbf{temp}^T\mathbf{temp}}$;
    $[\mathbf{Q}]_{:,j} := r_{last}\mathbf{temp}$;
    $proj := \mathbf{w}^T[\mathbf{Q}]_{:,j}$;
    $\mathbf{f} := \mathbf{f} - proj[\mathbf{Q}]_{:,j}$;
  end if
  $\mathbf{v} := \mathbf{f}^T\mathbf{Y}$;
  $\mathbf{idx}_i := \arg\max_{k=1,...,N}|\mathbf{v}_k|$;
end for
OUPUT $\mathbf{idx}$

determinar. Verifica-se também o tipo precisão necessário para este manter o mesmo desempenho. A precisão varia entre vírgula flutuante 64 e 32 bits, e fixa 64, 48 e 32 bits. Uma vez que as métricas SAM, SID e ED apresentam um padrão semelhante no seu comportamento, nas análises seguintes é apenas apresentada a métrica SAM.

3.2.1 Análise em função da SNR

Nesta análise realiza-se 100 experiências por cada imagem com SNR diferente, onde cada imagem processa-se por cinco métodos com precisão diferente. Estas imagens têm uma SNR de 40, 50 e infinito dB, 10000 píxeis, onde cada píxel é uma mistura de três assinaturas e as suas abundâncias geram-se a partir de uma distribuição Dirichlet com os seguintes parâmetros $\mu_{1..3} = 1/3$.

A Figura 3.6 apresenta a média e a variância da avaliação da SAM de cada tipo de precisão em função da SNR dos dados. Os resultados demonstram que o desempenho do método aumenta face ao aumento da SNR, com vírgula flutuante 64 e 32 bits, e fixa 64 e 48 bits. Em vírgula fixa 32-bit o desempenho do método piora, onde a média é da ordem dos 2º. Esta degradação deve-se principalmente à falta de precisão

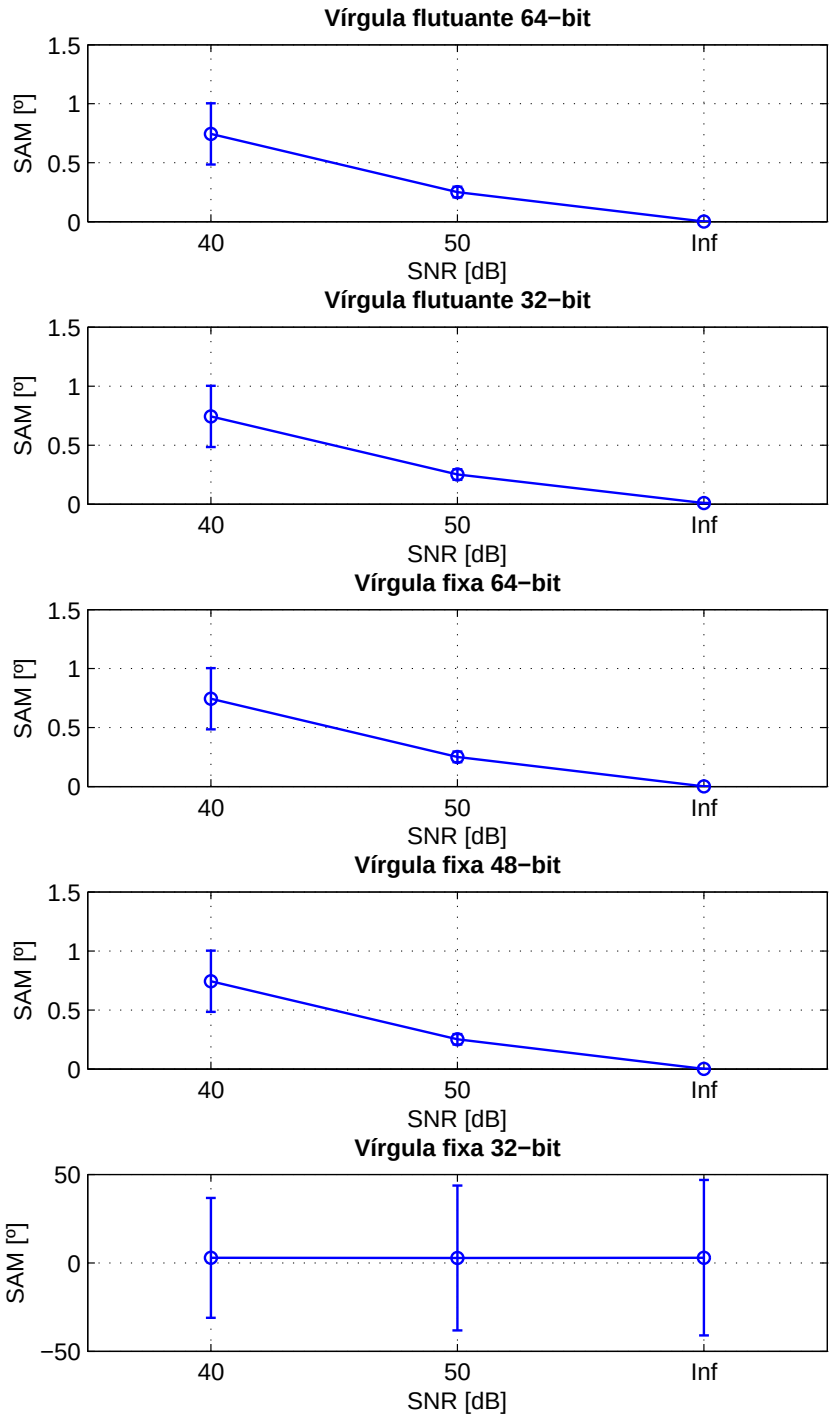
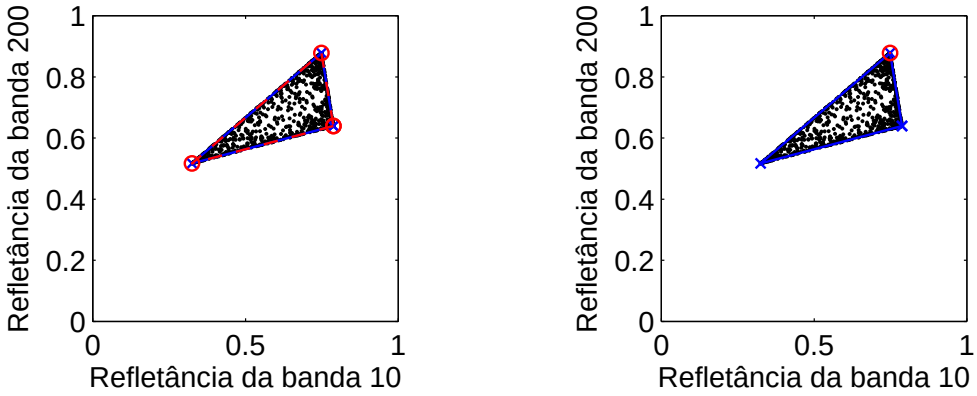


Figura 3.6: Média e variância dos resultados da avaliação da SAM das experiências em função da SNR dos dados com o método a operar em vírgula flutuante 64-bit, 32-bit e fixa 64-bit, 48-bit, 32-bit.

no cálculo dos vetores ortogonais Q, pois o produto interno entre esses vetores e o vetor f é maior que nos casos anteriores.

Quando o método opera com vírgula fixa 32-bit muitas vezes determina várias vezes o mesmo *endmember*. A Figura 3.7 apresenta o *simplex* dos dados, onde o eixo

das abcissas e das coordenadas representam a banda 110 e 190 respetivamente, com vírgula flutuante com precisão dupla (Figura 3.7a) e vírgula fixa com 32-bit (Figura 3.7b), onde à azul é representado o *simplex* correto e a vermelho o *simplex* determinado. No caso da precisão vírgula flutuante o *simplex* é determinado corretamente, mas em vírgula fixa este determina três vezes o mesmo *endmember*.



(a) Vírgula flutuante 64-bit (b) Vírgula fixa com 32 bits

Figura 3.7: *Simplex* projetado na banda 10 e 200 da experiência com vírgula flutuante 64-bit (a) e vírgula fixa 32-bit (b).

Uma vez que o desempenho do método otimizado diminui, quando este opera em vírgula fixa 32-bit, este tipo de precisão não é considerado na análise seguinte.

3.2.2 Análise em função do número de *endmembers*

Nesta análise são realizadas 100 experiências por cada imagem com número de assinaturas diferente, onde cada imagem é processada por quatro métodos com precisão diferente. Estas imagens têm 10000 píxeis, onde cada píxel é uma mistura de p assinaturas ($p = 3, 6, 9, 12, 15, 18$) e as suas abundâncias são geradas por uma distribuição Dirichlet com os seguintes parâmetros $\mu_{1..p} = 0, 1/p$ e uma SNR de 50 dB.

A Figura 3.8 apresenta a média e a variância da avaliação da SAM de cada tipo de precisão em função do número de *endmembers* a determinar. Os resultados demonstram que o desempenho do método é semelhante ao demonstrado anteriormente, mas em vírgula fixa 48-bit o seu desempenho piora.

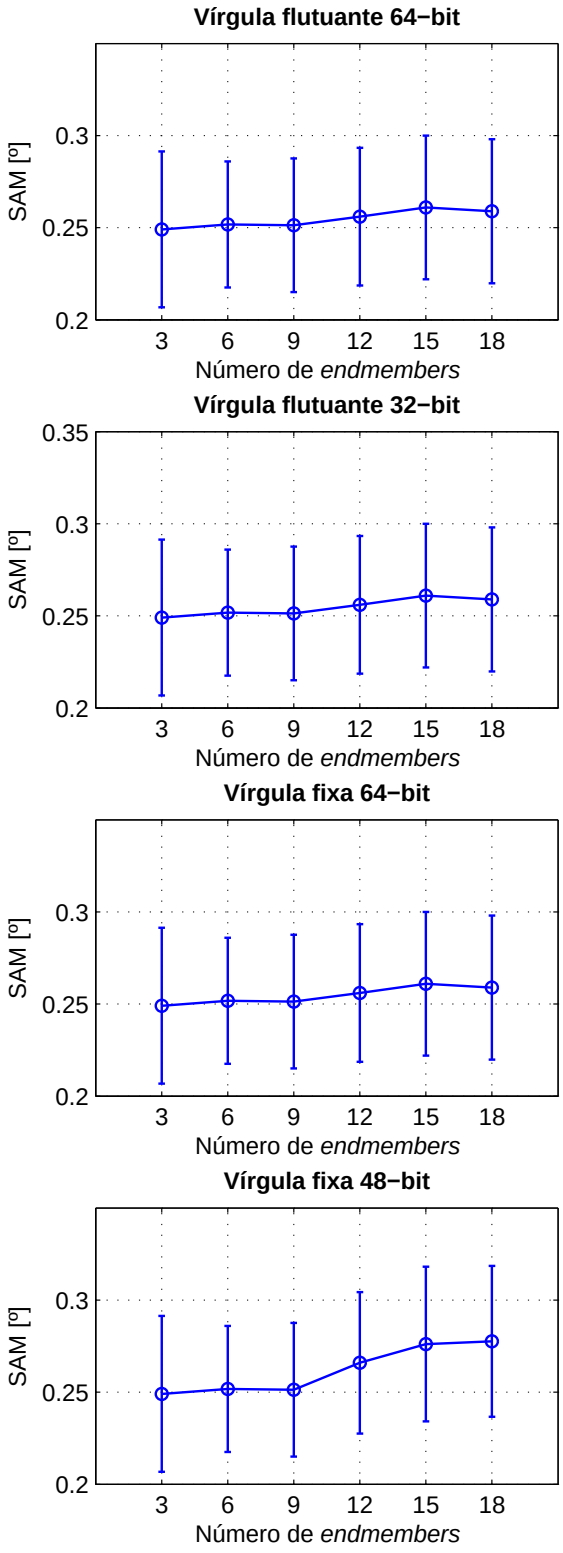


Figura 3.8: Média e variância dos resultados da avaliação da SAM das experiências em função do número de *endmembers* a determinar com o método a operar em vírgula flutuante 64-bit, 32-bit e fixa 64-bit, 48-bit.

4 Arquitetura

Com base no método VCA otimizado é projetada uma arquitetura em *hardware* que paraleliza o método e que pode ser adaptada em tempo de síntese às características de qualquer sensor. A arquitetura projetada é escalável, podendo-se definir o nível de paralelismo da arquitetura. As características da arquitetura que podem ser alteradas são o número de bandas espetrais a processar em paralelo, o número de *endmembers* a determinar e o número total de píxeis e bandas a processar. A primeira característica é definida pelo nível de paralelismo presente em alguns módulos da arquitetura e as restantes características são definidas pela lógica de controlo e pela dimensão das memórias que armazenam os dados necessários na arquitetura. Face aos resultados expostos no capítulo anterior, a arquitetura é projetada para operar com vírgula flutuante 32-bit.

4.1 Descrição geral

A arquitetura (ver Figura 4.1) é composta pelos seguintes módulos principais:

- **Memória** – armazena os dados de entrada e de saída do método, juntamente com todos os resultados intermédios durante a execução do método,
- **Gerador de números aleatórios** – gera números aleatórios de vírgula flutuante com precisão simples,
- **Conversor de inteiro para vírgula flutuante** – converte um número inteiro para um número com vírgula flutuante com precisão simples,
- **Produto interno** – calcula o produto interno entre dois vetores,

- **Multiplicador e subtrator** – calcula a multiplicação entre dois escalares ou multiplica dois escalares e subtrai o resultado por outro escalar,
- **Recíproco da raiz quadrada** – calcula o recíproco da raiz quadrada de um escalar,
- **Argumento máximo** – indica o argumento máximo de um vetor.

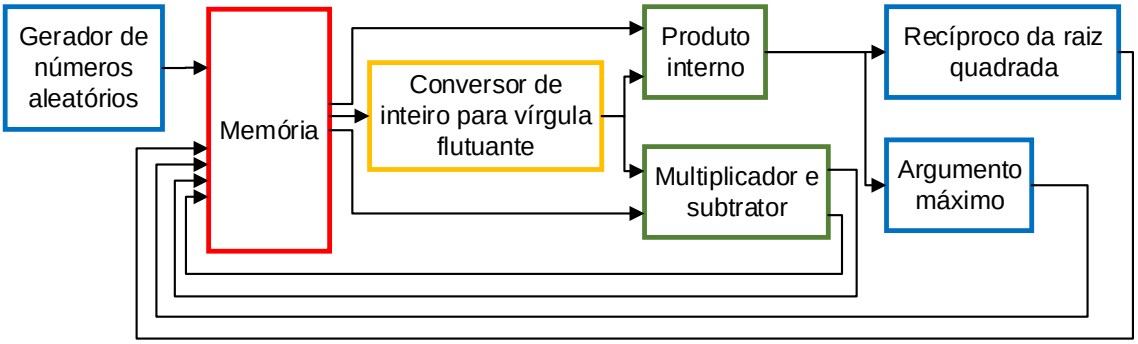


Figura 4.1: Diagrama geral da arquitetura em *hardware* do método VCA otimizado.

Os módulos da arquitetura são controlados por uma máquina de estados que controla de acordo com o fluxo de dados do método, como se mostra na Figura 4.2. Esta realiza uma ou várias linhas do Algoritmo 2 e é composta pelos seguintes estados principais:

- **St_start** – inicia os contadores it e j e gera o vetor aleatório ($\mathbf{w}$) com a utilização do módulo *Gerador de números aleatórios* (linha 2-3),
- **St_qr1** – determina os coeficientes $\mathbf{r}$ da decomposição QR com a utilização do módulo *Produto interno* (linha 10),
- **St_qr2** – determina um vetor auxiliar $\mathbf{temp}$ para determinar o novo vetor ortogonal da base $\mathbf{Q}$ com a utilização do módulo *Multiplicador e subtrator* (linha 11),
- **St_qr3** – determina o último coeficiente r_{last} da decomposição com a utilização dos módulos *Produto interno* e *Recíproco da raiz quadrada* (linha 13),
- **St_qr4** – determina o novo vetor ortogonal da base $\mathbf{Q}$ com a utilização do módulo *Multiplicador e subtrator* (linha 14),

- **St_f1** – determina um escalar auxiliar $proj$ para determinar a nova direção ortogonal $\mathbf{f}$ com a utilização do módulo *Multiplicador e subtrator* (linha 15),
- **St_f2** – determina a direção ortogonal $\mathbf{f}$ com a utilização do módulo *Multiplicador e subtrator* (linha 16),
- **St_v** – projeta os dados $\mathbf{Y}$ na direção ortogonal $\mathbf{f}$ e determina o argumento máximo do novo *endmember* com utilização dos módulos *Produto interno* e *Argumento máximo* (linha 18-19),
- **St_it_end** – assinala o fim de uma iteração do método e verifica se todos os *endmembers* já foram determinados (linha 5),
- **St_it_inc** – incrementa os contadores it e j (linha 5 e 9),
- **St_done** – assinala o fim do método.

4.1.1 Memória

O módulo *Memória* é composto por nove componentes de memória designados por:

- **Memória Y** – armazena os dados a serem processados pelo método, vindo do protocolo de entrada definido,
- **Memória w** – armazena os números aleatórios gerados pelo módulo *Gerador de números aleatórios*,
- **Memória f** – armazena os dados correspondentes à direção ortogonal produzidos pelo módulo *Multiplicador e subtrator*,
- **Memória temp** – armazena os dados intermédios da decomposição QR produzidos pelo módulo *Multiplicador e subtrator*,
- **Memória Q** – armazena os vetores ortogonais da decomposição QR produzidos pelo módulo *Multiplicador e subtrator*,
- **Memória idx** – armazena os índices dos píxeis determinados como *endmembers* produzidos pelo módulo *Comparador*,

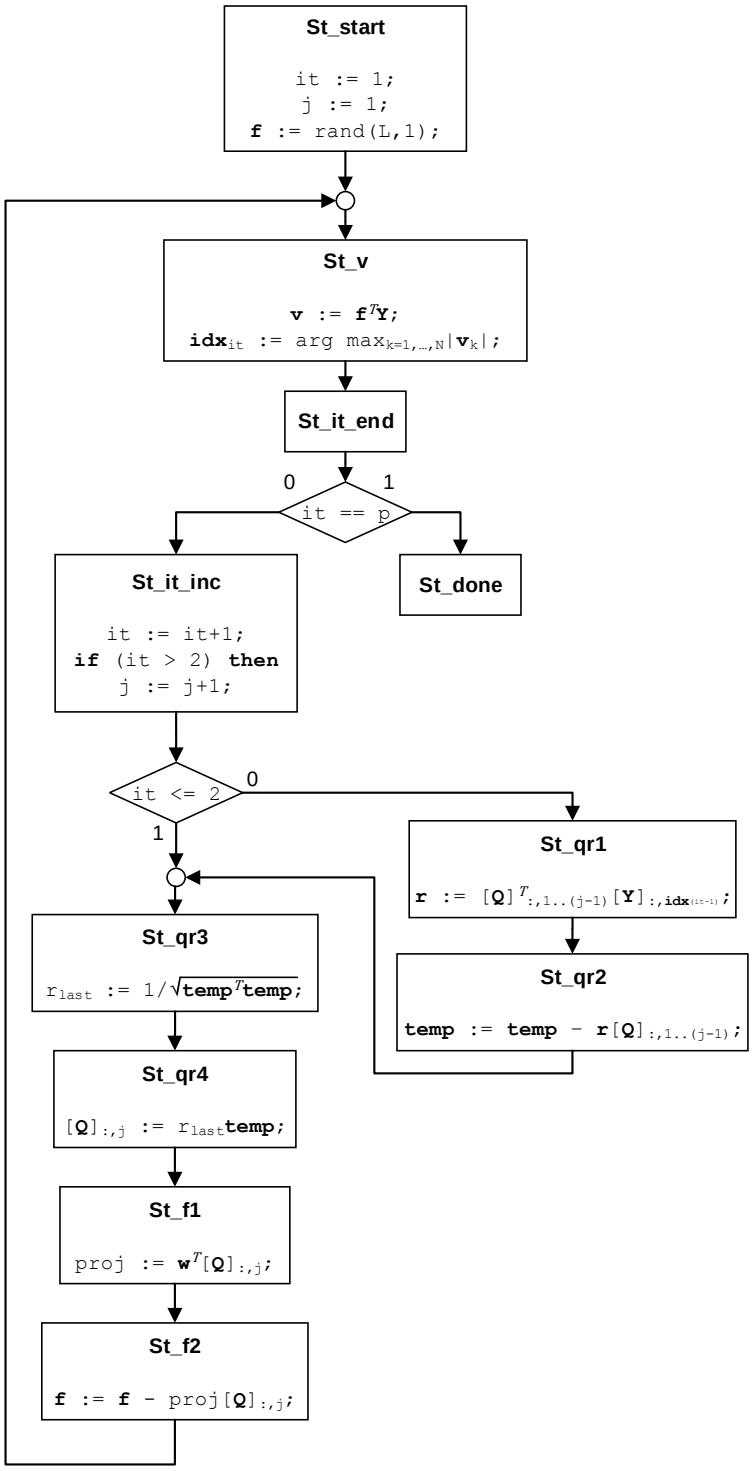


Figura 4.2: Máquina de estados da arquitetura em *hardware* do método VCA otimizado.

- **Lista r** – armazena os coeficientes R da decomposição QR produzidos pelo módulo *Produto interno*,

- **Registo $\mathbf{r}_{last}$** – armazena o último coeficiente R da decomposição QR produzido pelo módulo *Recíproco da raiz quadrada*,

- **Registo proj** – armazena um resultado intermédio da decomposição QR produzido pelo módulo *Produto interno*.

A Figura 4.3 apresenta o diagrama geral do módulo com as respetivas ligações de entrada de cada componente.

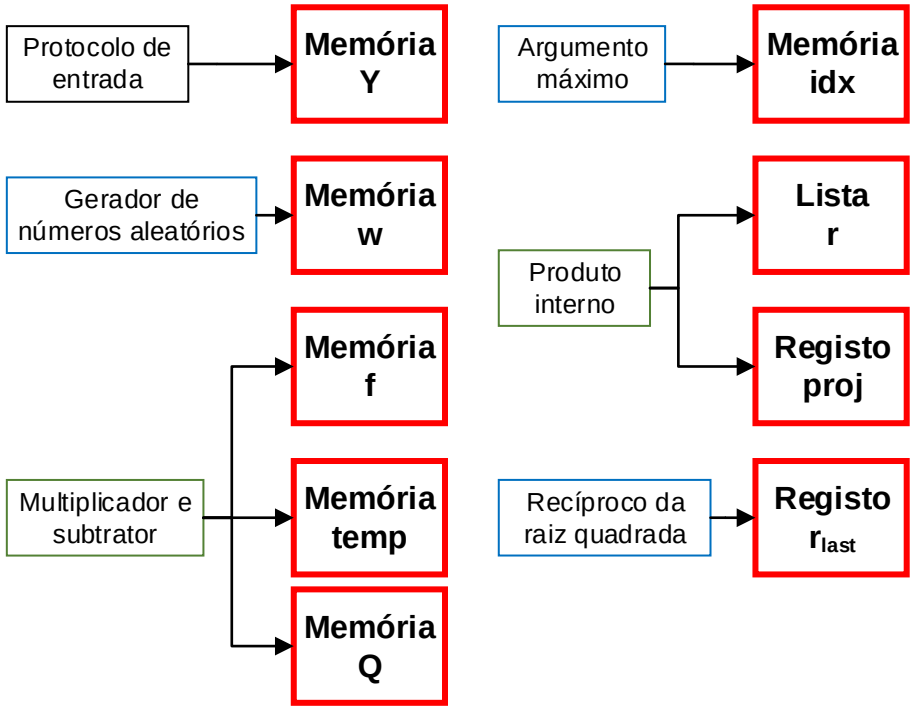


Figura 4.3: Diagrama geral do módulo *Memória* com as respetivas ligações de entrada.

4.1.2 Gerador de números aleatórios

O módulo *Gerador de números aleatórios* é usado para gerar números de vírgula flutuante com precisão simples. Os números produzidos pelo módulo são armazenados no componente *Memória w*. A Figura 4.4 apresenta o diagrama geral do módulo com a respetiva ligação de saída. O módulo é composto por um componente que gera números inteiros aleatórios com uma distribuição de Mersenne twister [57] e um outro que converte cada número gerado para vírgula flutuante.

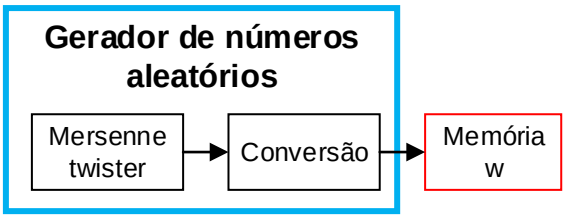


Figura 4.4: Diagrama geral do módulo *Gerador de números aleatórios* com a respetiva ligação de saída.

Esta conversão distribui 16 bits do número gerado pela mantissa, sendo o expoente fixo como se mostra na Figura 4.5. Para gerar mais números em paralelo, distribui-se outro conjunto de bits pelas respetivas mantissas. Na Figura 4.6 apresenta-se o histograma a partir de 180.000 amostras recolhidas do módulo, com média 1,5 e variância 0,08. Com base no histograma apresentado conclui-se que o módulo segue uma distribuição uniforme entre $[1; 2]$.

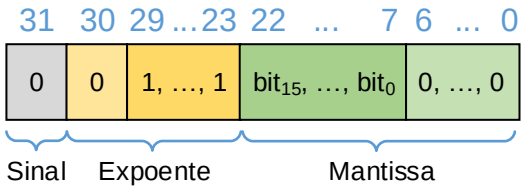


Figura 4.5: Geração de um número de vírgula flutuante com precisão simples a partir de um inteiro.

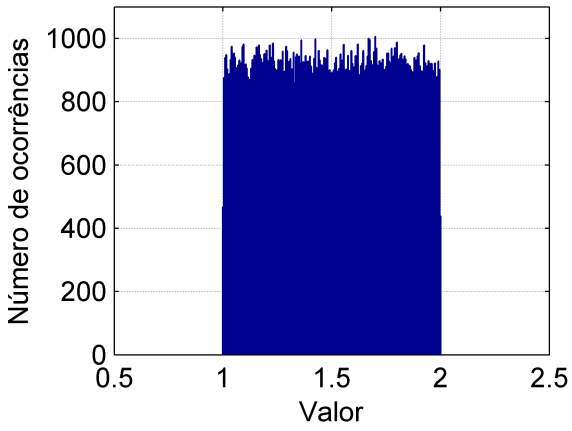


Figura 4.6: Histograma dos valores gerados pelo componente *Conversão*.

4.1.3 Conversor de inteiro para vírgula flutuante

O módulo *Conversor de inteiro para vírgula flutuante* é usado para converter os dados de saída da *Memória Y* para números de vírgula flutuante 32-bit. Estes dados depois são utilizados pelos módulos *Produto interno* e *Multiplicador e subtrator*. Este módulo é gerado a partir do Xilinx LogiCORE IP CORE Floating point [58]. A Figura 4.7 apresenta o diagrama geral do módulo com as respetivas ligações de entrada e de saída.

Figura 4.7: Diagrama geral do módulo *Conversor de inteiro para vírgula flutuante* com as respetivas ligações de entrada e saída.

4.1.4 Recíproco da raiz quadrada

O módulo *Recíproco da raiz quadrada* é usado para calcular o último coeficiente R da decomposição QR, a partir do resultado do módulo *Produto interno* e armazena o resultado no componente *Registo r*. Este módulo é gerado a partir do Xilinx LogiCORE IP CORE Floating point [58]. A Figura 4.8 apresenta o diagrama geral do módulo com as respetivas ligações de entrada e de saída.

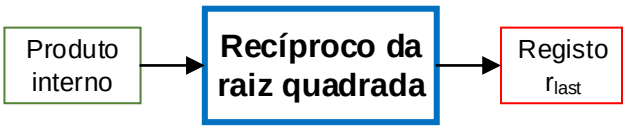


Figura 4.8: Diagrama geral do módulo *Recíproco da raiz quadrada* com as respetivas ligações de entrada e de saída.

4.1.5 Argumento máximo

O módulo *Argumento máximo* a cada iteração indica o índice do vetor que corresponde ao novo *endmember*. Este é constituído por um contador que indica o índice do vetor atual, um registo que indica o argumento máximo da iteração atual, um comparador de vírgula flutuante com precisão simples que indica qual o maior valor entre dois e um registo que indicia qual o maior valor da iteração atual. O comparador é gerado a partir do Xilinx LogiCORE IP CORE Floating point [58].

Por cada resultado produzido pelo módulo *Produto interno*, este é comparado com o maior valor até então produzido por esse módulo nessa iteração. Caso este seja maior, este é guardado num registo e é guardado o número do contador num registo. O contador é incrementado a cada resultado produzido. A Figura 4.9 apresenta o diagrama geral do módulo e as respetivas ligações de entrada e de saída.

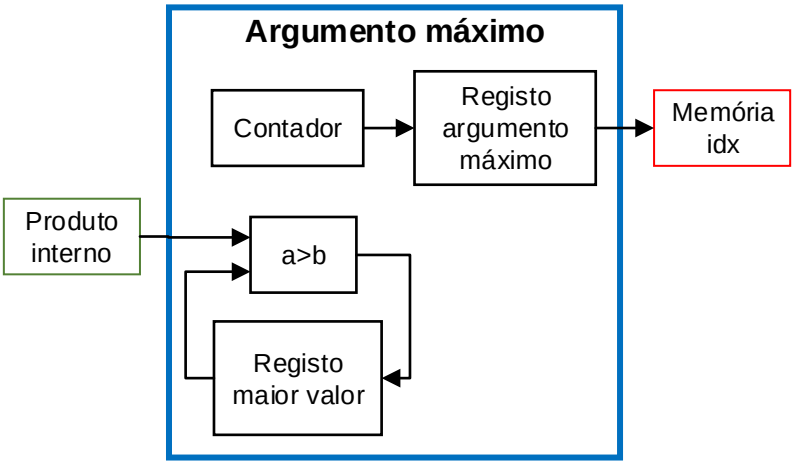


Figura 4.9: Diagrama geral do módulo *Comparador* com as respetivas ligações de entrada e de saída.

4.1.6 Multiplicador e subtrator

O módulo *Multiplicador e subtrator* é usado para multiplicar ou multiplicar e subtrair elementos de dois vetores. Cada entrada do módulo está multiplexada (MUX), pois os dados provêm de diferentes componentes de memória. O resultado da multiplicação é armazenado no componente *Memória Q* e o da multiplicação e subtração é armazenado nos componentes *Memória f* e *Memória temp*. A Figura 4.10 apresenta

o diagrama geral do módulo com as respetivas ligações de entrada e de saída. O módulo *Conversor de inteiro para vírgula flutuante* é representado pelo nome *int2float*.

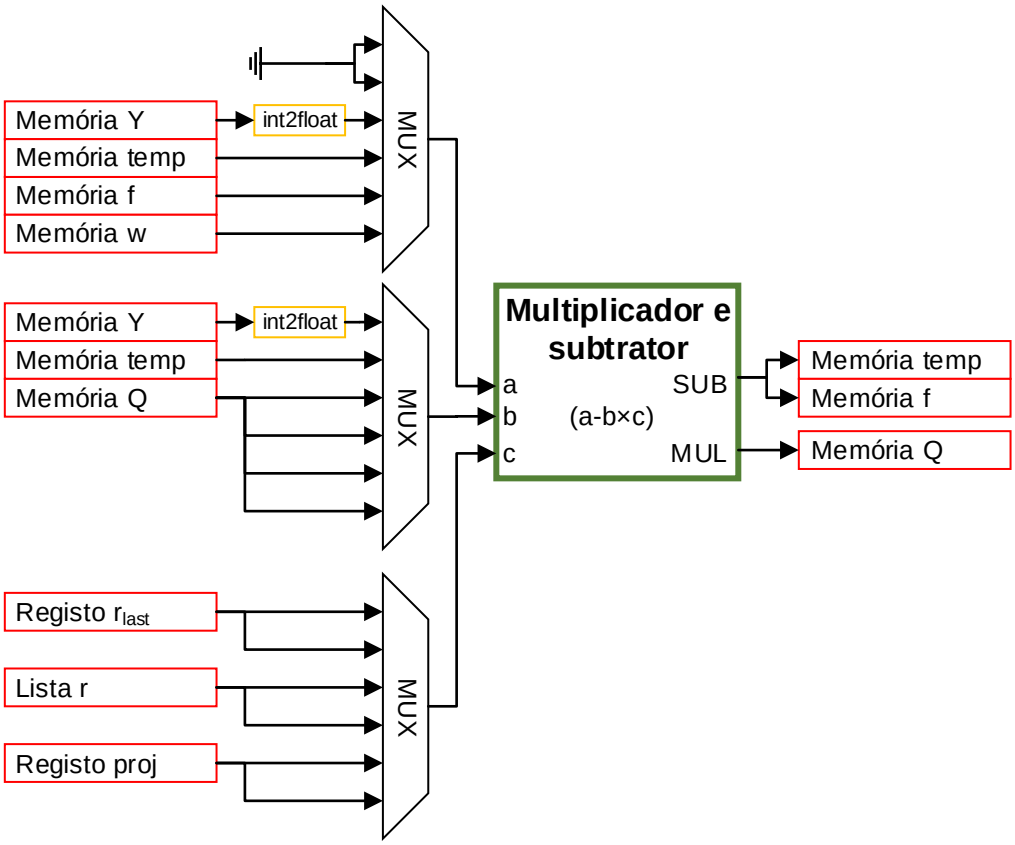


Figura 4.10: Diagrama geral do módulo *Multiplicador e subtrator* com as respetivas ligações de entrada e de saída.

Este módulo é escalável e consegue processar i elementos em paralelo, sendo este composto por i multiplicadores (MUL) e i subtratores (SUB). Cada multiplicador e subtrator é gerado a partir do Xilinx LogiCORE IP CORE Floating point [58]. A Figura 4.11 apresenta o diagrama com a composição do módulo.

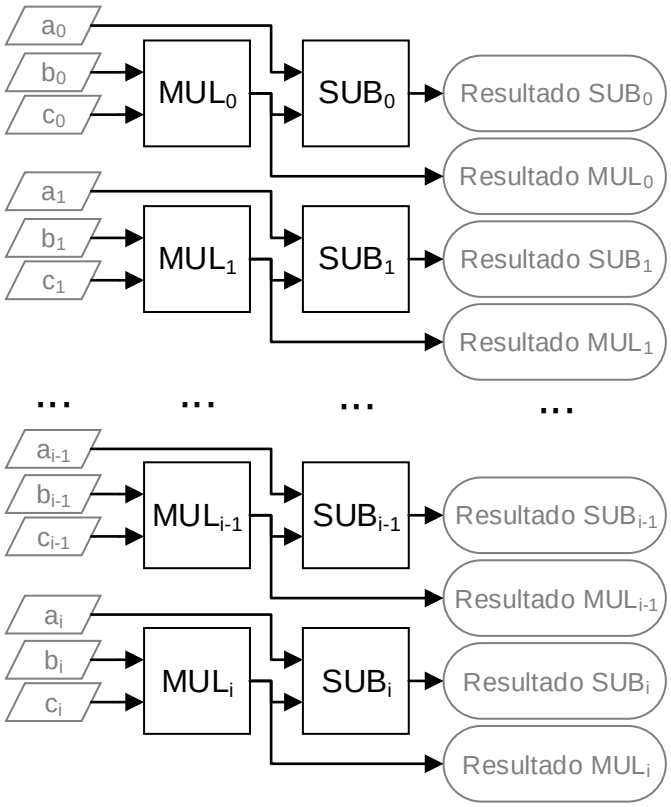


Figura 4.11: Diagrama com a composição do módulo *Multiplicador e subtrator*.

4.1.7 Produto interno

O módulo *Produto interno* é usado para realizar o produto entre dois vetores. Cada entrada do módulo também está multiplexada (MUX), pois os dados provêm de diferentes componentes de memória. O resultado do módulo é armazenado no componente *Lista r*, *Registo proj* e utilizado pelos módulos *Recíproco da raiz quadrada* e *Argumento máximo*. A Figura 4.12 apresenta o diagrama geral do módulo com as respetivas ligações de entrada e de saída. O módulo *Conversor de inteiro para vírgula flutuante* é representado pelo nome *int2float*.

Este módulo é escalável e consegue processar i elementos em paralelo, sendo este composto por i multiplicadores (MUL), i acumuladores (ACC) e por uma árvore de somadores (ADD), para somar o resultado produzido por cada acumulador e assim produzir o resultado do produto interno entre os vetores. Cada multiplicador e somador é gerado a partir do Xilinx LogiCORE IP CORE Floating point [58]. A Figura 4.13 apresenta o diagrama com a composição do módulo.

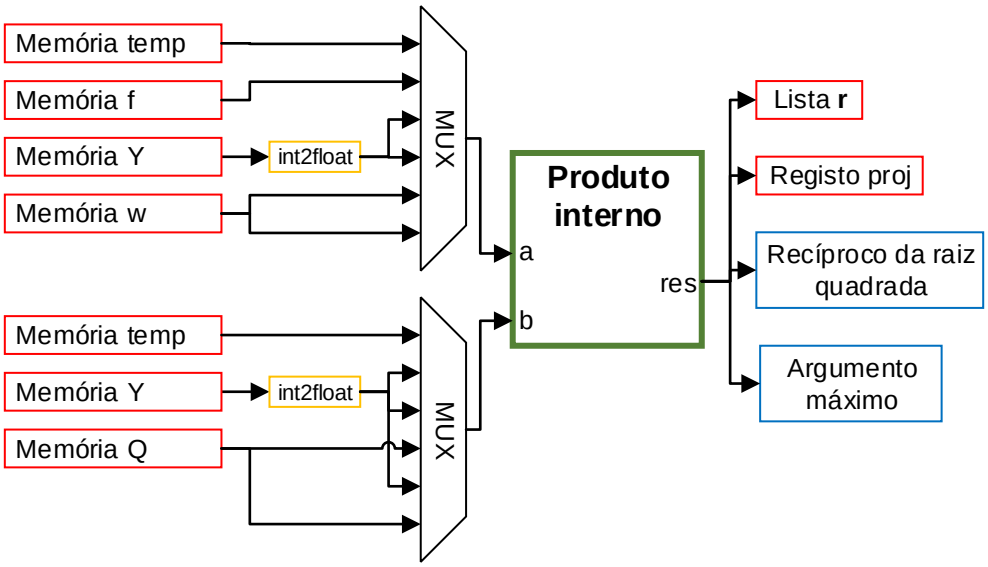


Figura 4.12: Diagrama geral do módulo *Produto interno* com as respetivas ligações de entrada e de saída.

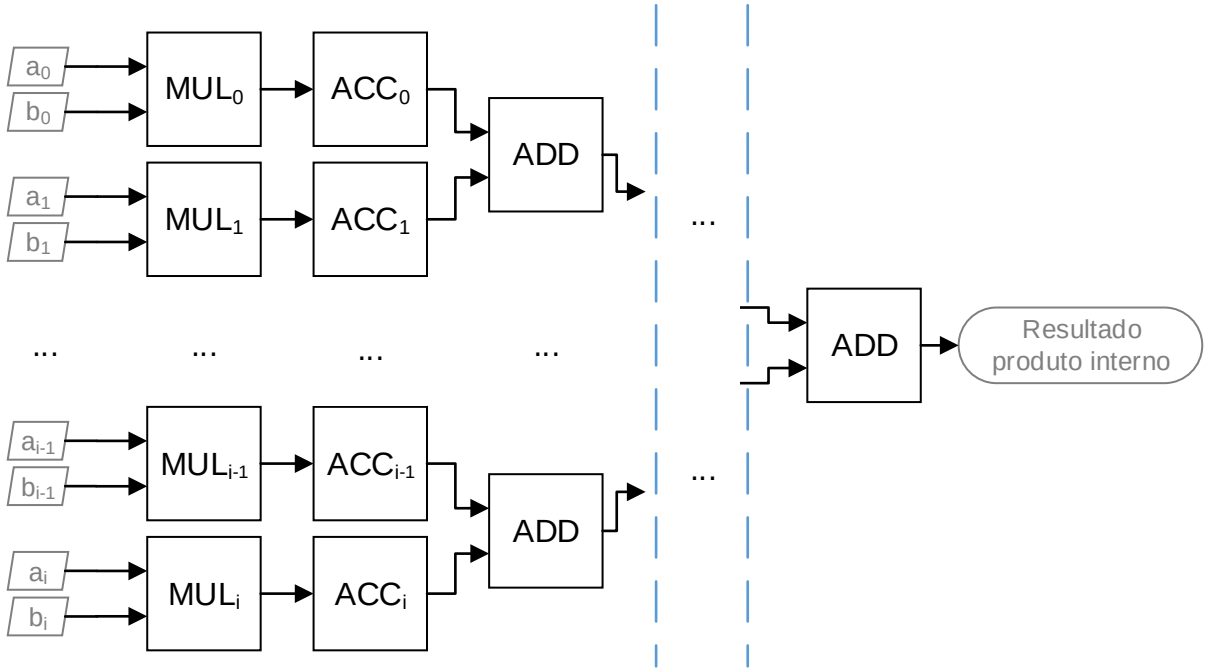


Figura 4.13: Diagrama com a composição do módulo *Produto interno*.

O componente acumulador é projetado por forma a continuar a operar sem ser interrompido o fluxo de dados. Este é composto por um somador com três de latência (Lat = 3), um registo que liga a saída do somador a uma das suas entradas, um

conjunto de registos que produzem atrasos e uma árvore de somadores, onde cada somador tem seis de latência (Lat = 6), como se mostra na Figura 4.14. O primeiro

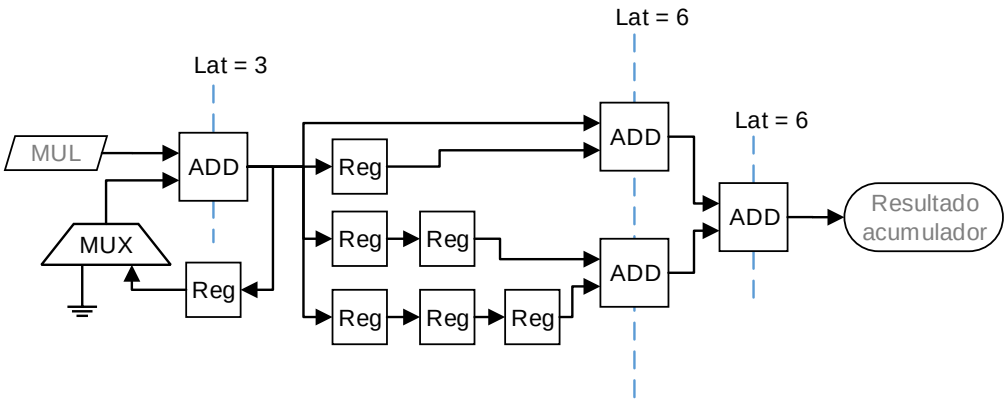


Figura 4.14: Diagrama com a composição do componente *acumulador*.

somador soma os valores produzidos pelo multiplicador ($mul_{0..L}$) com zero ou com a própria saída atrasada por uma latência, como se mostra na Figura 4.15. Os resultados desse somador passam pelo conjunto de registos e depois são somados pela árvore de somadores. Ao fim de L ciclos é colocado zero numa das entradas do somador por forma que a nova acumualação não seja afetada pela anterior.

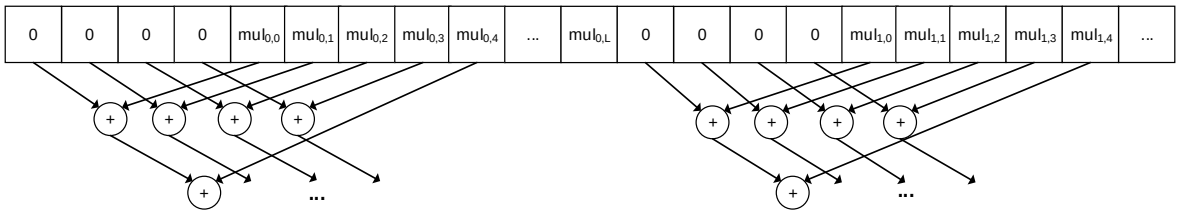


Figura 4.15: Esquema de saída do primeiro somador do componente *acumulador*.

4.2 Análise teórica da arquitetura

A análise teórica à arquitetura verifica o número de ciclos de relógio de cada estado, consoante o número de bandas (par_L) e número de píxeis (par_N) que esta processa em paralelo. Um ciclo de relógio representa uma multiplicação seguida por uma acumulação/subtração. A Tabela 4.1 apresenta a expressão do número de ciclos dos estados $\mathrm{St\_qr1}$, $\mathrm{St\_qr2}$, $\mathrm{St\_qr3}$, $\mathrm{St\_qr4}$, $\mathrm{St\_f1}$, $\mathrm{St\_f2}$ e $\mathrm{St\_v}$, onde L representa

o número de bandas de um píxel N o número de píxeis, it o número de iterações, par_L representa o número de bandas processadas em paralelo em todos os estados e par_N representa o número de píxeis processados em paralelo no estado **St_v**. O somatório nos estados **St_qr1** e **St_qr2** são progressões geométricas com expressão conhecida.

Estado	**Número de ciclos**
St_qr1	$L/par_L \times \sum_{j=1}^{it-2} j$
St_qr2	$L/par_L \times \sum_{j=1}^{it-2} j$
St_qr3	$L/par_L \times (it-1)$
St_qr4	$L/par_L \times (it-1)$
St_f1	$L/par_L \times (it-1)$
St_f2	$L/par_L \times (it-1)$
St_v	$L/par_L \times N/par_N \times it$

Tabela 4.1: Descrição da expressão do número de ciclos dos estados St_qr1, St_qr2, St_qr3, St_qr4, St_f1, St_f2 e St_v.

A Tabela 4.2 apresenta um exemplo do número de ciclos dos estados St_qr1, St_qr2, St_qr3, St_qr4, St_f1, St_f2, St_v e número total de ciclos, supondo oito iterações ($it = 8$), 614 píxeis ($N = 614$), 224 bandas espetrais ($L = 224$) com vários níveis de paralelismo. O número de iterações depende do número de *endmembers* a determinar (p) sendo este exemplo para $p = 8$. No entanto pode-se ajustar para qualquer outro valor entre 0 e 8.

Paralelismo		**Número de ciclos**							
		Estado							
par_L	par_N	St_qr1	St_qr2	St_qr3	St_qr4	St_f1	St_f2	St_v	Total
1	1	4 704	4 704	1 568	1 568	1 568	1 568	1 100 288	1 115 968
2	1	2 352	2 352	784	784	784	784	550 144	**557 984**
1	2	4 704	4 704	1 568	1 568	1 568	1 568	550 144	565 824
2	2	2 352	2 352	784	784	784	784	275 072	282 912
4	1	1 176	1 176	392	392	392	392	275 072	**278 992**
1	4	4 704	4 704	1 568	1 568	1 568	1 568	275 072	290 752
4	2	1 176	1 176	392	392	392	392	137 536	141 456
2	4	2 352	2 352	784	784	784	784	137 536	145 376
8	1	588	588	196	196	196	196	137 536	**139 496**
1	8	4 704	4 704	1 568	1 568	1 568	1 568	137 536	153 216

Tabela 4.2: Descrição do número de ciclos dos estados St_qr1, St_qr2, St_qr3, St_qr4, St_f1, St_f2, St_v e número total de ciclos.

Com base na Tabela 4.2 verifica-se que o estado **St_v** é o que tem o número mais elevado de ciclos e representa mais de 90% do número de total de ciclos. Neste estado realiza-se a projeção dos dados, onde o vetor $\mathbf{f}$ de dimensão L é multiplicado pela matriz $\mathbf{Y}$ de dimensão $L \times N$. Também verifica-se que é mais vantajoso processar em paralelo mais bandas do que vários píxeis, pois aumentar o paralelismo no módulos *Produto interno* e *Multiplicado e subtrator* beneficia todos os estados, do que ter vários módulos *Produto interno* e apenas beneficiar o estado **St_v**.

A Tabela 4.3 apresenta o tempo total de execução das arquiteturas com $par_L = 1$, $par_L = 2$, $par_L = 4$, $par_L = 8$ e todas com $par_N = 1$ para várias frequências.

Frequência [MHz]	**Tempo de execução** [ms]				
	par_L	1	2	4	8
	par_N	1	1	1	1
50		22,32	11,16	5,58	2,79
60		18,60	9,30	4,65	2,32
70		15,94	7,97	3,99	1,99
80		13,95	6,97	3,49	1,74
90		12,40	6,20	3,10	1,55
100		11,16	5,58	2,79	1,40
110		10,15	5,07	2,54	1,27
120		9,30	4,65	2,32	1,16
130		8,58	4,29	2,15	1,07
140		7,97	3,99	1,99	1,00
150		7,44	3,72	1,86	0,93

Tabela 4.3: Descrição do tempo de execução total teórico das arquiteturas com $par_L = 1$, $par_L = 2$, $par_L = 4$, $par_L = 8$ e todas com $par_N = 1$ para várias frequências.

5 | Implementação e resultados

A arquitetura é descrita na linguagem de descrição de *hardware* VHDL e implementada na Xilinx Zynq Zedboard com o integrado XC7Z020 *System on Chip* (SoC). Este FPGA é composto por um Dual ARM® Cortex™-A9 e uma área reconfigurável equivalente a um ARTIX®-7. A arquitetura é desenvolvida nas ferramentas Xilinx ISE, PlanAhead, Platform Studio e Software Development Kit.

Os componentes de memória, multiplicadores, somadores são gerados a partir da ferramenta Xilinx Core Generator. Os componentes de vírgula flutuante são gerados com o Xilinx LogiCORE IP CORE Floating point [58] e as memórias geradas com o Xilinx LogiCORE IP Block Memory Generator [59].

5.1 Implementação da arquitetura

A arquitetura em *hardware* é implementada para o sensor AVIRIS [3]. Este sensor adquire 512 píxeis em 8,3 ms [19], onde cada píxel é um vetor de 224 elementos e cada elemento é quantificado com 16 bits [60]. A arquitetura opera em vírgula flutuante 32-bit, determinar oito *endmembers* e processar 614 píxeis de 224 bandas espetrais. Com base nas conclusões apresentadas no capítulo anterior, a arquitetura é implementada quatro vezes com as seguintes características de paralelismo:

- **Arquitetura 1 –** $par_L = 1$ e $par_N = 1$,
- **Arquitetura 2 –** $par_L = 2$ e $par_N = 1$,
- **Arquitetura 3 –** $par_L = 4$ e $par_N = 1$,
- **Arquitetura 4 –** $par_L = 8$ e $par_N = 1$.

No entanto, a arquitetura pode processar menos píxeis e/ou bandas, bastando simplesmente preencher com zeros os píxeis e/ou bandas não utilizados.

A Tabela 5.1 apresenta os respetivos recursos utilizados de todas implementações.

Recurso	**Arquitetura**			
	1	**2**	**3**	**4**
Register	4 188 (4%)	6 131 (6%)	10 431 (10%)	18 758 (18%)
LUT	3 872 (7%)	6 753 (13%)	11 159 (21%)	20 236 (38%)
Slice	1 916 (14%)	3 077 (23%)	4 525 (34%)	7 729 (58%)
RAM36Kb	76 (54%)	72 (51%)	76 (54%)	82 (59%)
DSP48	21 (10%)	35 (16%)	63 (29%)	119 (55%)

Tabela 5.1: Resumo dos recursos utilizados do FPGA das arquiteturas implementadas do algoritmo VCA.

A Tabela 5.2 apresenta a frequência máxima, o número total de ciclos teórico e prático e o tempo total de execução teórico e prático das respetivas implementações. A frequência máxima em todas as arquiteturas é definida pela ligação entre a lógica de

Características	**Arquitetura**			
	1	**2**	**3**	**4**
Frequência máxima [MHz]	120	100	100	90
Número total de ciclos teórico	1 115 968	557 984	278 992	139 496
Número total de ciclos prático	1 117 139	559 335	280 523	141 207
Tempo total de execução teórico [ms]	9, 30	5, 58	2, 79	1, 55
Tempo total de execução prático [ms]	9, 31	5, 60	2, 81	1, 57

Tabela 5.2: Resumo do tempo de execução das arquiteturas implementadas do algoritmo VCA.

controlo e a memória que armazena os dados (mais de 45% das RAMs disponíveis). Esta ligação é exigente, pois cria um constrangimento físico para ligar com todas essas RAMs.

Para o sensor AVIRIS que adquire 512 píxeis com 224 bandas em 8,3 ms, todas as arquiteturas (à exceção da primeira) são adequadas para processamento em tempo-real abordo.

5.2 Avaliação de resultados

O *hardware* implementado é testado com uma imagem hiperespetral real adquirida pelo sensor AVIRIS sobre o distrito Cuprite do estado de Nevada nos Estados Unidos

da América. Esta imagem é sobretudo uma zona mineral, bem descriminada, e expõe vários minerais de interesse representados na biblioteca digital USGS [47]. Desta é apenas utilizada uma subimagem de 250$\times$191 pixeis e das 224 bandas espetrais disponíveis são removidas as bandas de absorção de água (96-130 e 147-171) e também as bandas com baixo SNR (1-4 e 221-224), deixando um total de 156 bandas espetrais para as experiências realizadas. A Figura 5.1 apresenta a banda 30 (comprimento de onda $\lambda = 667\ nm$) da subimagem considerada.

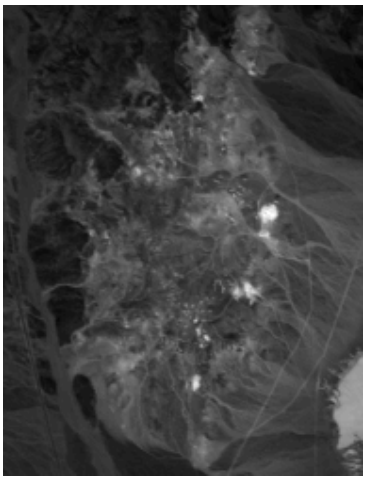


Figura 5.1: Banda 30 da subimagem da imagem da Cuprite.

De forma a processar os 47750 píxeis da subimagem, a arquitetura processa blocos de 614 píxeis de cada vez. Os *endmembers* determinados na última iteração são agrupados ao próximo bloco de píxeis. Neste processo é utilizada a memória DDR, disponível na placa, para armazenar o *dataset* e um dos processadores ARM, disponível no SoC/FPGA, para gerir o processo de envio de dados e recolha de resultados através do protocolo AXI4, como mostra a Figura 5.2.

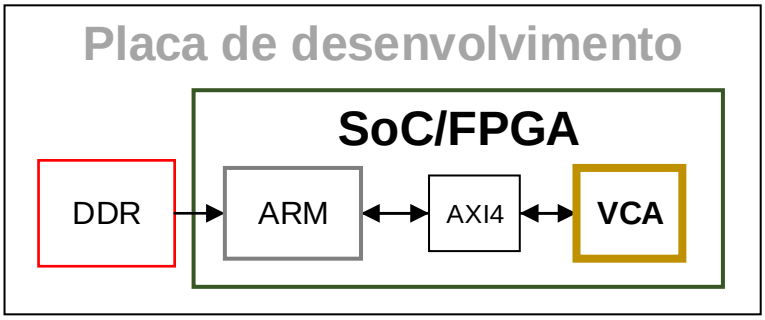


Figura 5.2: Processo de teste da arquitetura.

A Figura 5.3 apresenta a comparação entre a assinatura determinada (tracejado) com a refletância laboratorial mais próxima (linha) dos materias Alunite, Andradite, Buddingtonite, Kaolinite, Montmorillonite, Muscovite, Nontronite e Pyrope.

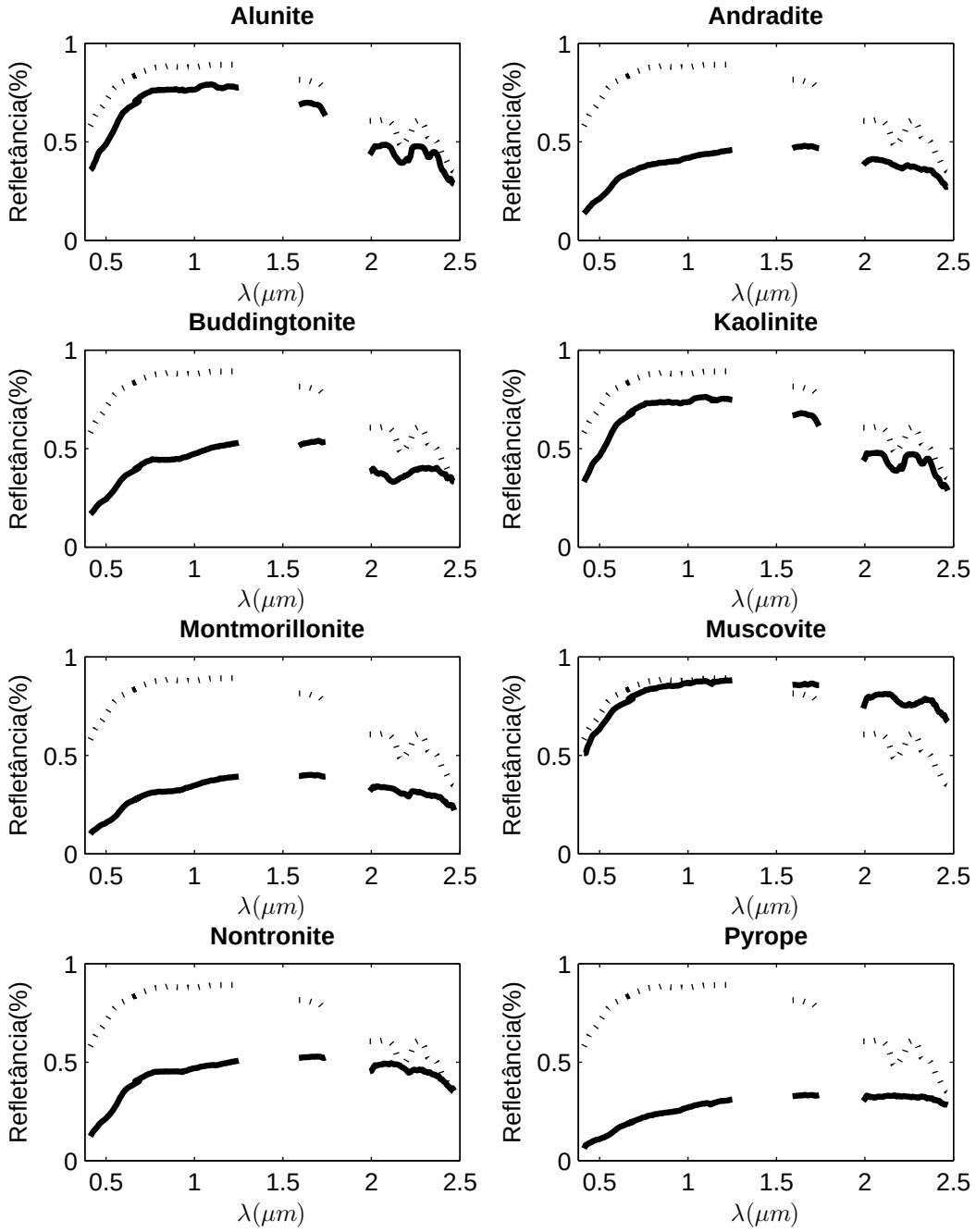


Figura 5.3: Comparação entre a assinatura determinada (tracejado) com a refletância laboratorial (linha) dos materiais Alunite, Andradite, Buddingtonite, Kaolinite, Montmorillonite, Muscovite, Nontronite e Pyrope.

A Tabela 5.3 apresenta a respetiva avaliação da SAM, SID e ED entre as assinaturas determinadas com a refletância de laboratório mais próxima para as substâncias. Para efeitos comparativos também são apresentados os resultados da subimagem

com redução de dados. Os resultados com redução de dados demonstram um melhor

Substâncias	SAM		SID		ED	
	Redução de dados					
	sim	não	sim	não	sim	não
Alunite	3,72	3,72	0,0055	0,0055	0,092	0,092
Andradite	3,26	4,27	0,0056	0,0074	0,080	0,105
Buddingtonite	3,12	3,41	0,0052	0,0053	0,077	0,084
Kaolinite	4,50	4,74	0,0103	0,0093	0,129	0,117
Montmorillonite	2,73	3,00	0,0030	0,0043	0,067	0,074
Muscovite	3,53	3,99	0,0061	0,0055	0,087	0,099
Nontronite	3,49	4,42	0,0074	0,0103	0,086	0,109
Pyrope	2,45	2,49	0,0037	0,0026	0,061	0,062

Tabela 5.3: Resultados SAM, SID e ED entre os *endmembers* determinados no *hardware* e refletâncias laboratoriais.

desempenho, pois a SNR dos dados aumenta e o que permite ao método determinar melhor os píxies puros presentes de cada substância. Estes resultados são semelhantes aos publicados no trabalho original do VCA [40].

6 | Conclusões e trabalho futuro

6.1 Conclusões

O presente trabalho propõe uma arquitetura num *field-programmable gate array* (FPGA), que paraleliza o método *vertex component analysis* (VCA) de separação de dados hiperespetrais. Este trabalho é desenvolvido na placa ZedBoard que contém um Xilinx Zynq®-7000 XC7Z020.

O método VCA iterativamente projeta os píxeis numa direção ortogonal no subespaço gerado pelas assinaturas espetrais já determinadas. A nova assinatura corresponde ao extremo dessa projeção e este processo repete-se até serem determinadas todas as assinaturas.

O desempenho do método é analisado em função de três parâmetros: número de píxeis da imagem, relação sinal ruído (*signal-to-noise ratio* - SNR) da imagem e número de assinaturas espetrais a determinar, sendo esta análise realizada com dados sintéticos. É concluído que o desempenho do método mantém-se constante em função do número de píxeis e número de assinaturas. O desempenho do método em função da SNR aumenta face ao aumento da SNR dos dados.

O método é otimizado com o objetivo de reduzir o peso e complexidade computacional do método original. A tarefa mais complexa deste é o cálculo da direção ortogonal, pois esta requer calcular uma matriz pseudo-inversa. No método original essa matriz é calculada através da decomposição em valores singulares (*singular value decomposition* - SVD). Neste trabalho propõe-se simplificar o cálculo através de uma decomposição QR com base no processo de Gram-Schimdt clássico. O algoritmo utilizado nesta decomposição é incremental, ou seja, este reutiliza os vetores ortogonais $\mathbf{Q}$ anteriormente calculados e a cada iteração apenas calcula o novo vetor ortogonal a incrementar na matriz $\mathbf{Q}$.

O desempenho do método otimizado é também analisado em função de dois parâmetros: SNR da imagem e número de assinaturas a determinar. Em junção destes dois parâmetros é também verificado o tipo de precisão necessária para manter o mesmo desempenho, entre virgula flutuante e fixa. Conclui-se que o método necessita de pelo menos vírgula flutuante 32-bit ou fixa 48-bit.

Com base no método otimizado é projetada uma arquitetura que paraleliza o método e que pode ser adaptada em tempo de síntese às características de qualquer sensor. A arquitetura é escalável, podendo-se definir o número de bandas espetrais e/ou píxeis a processar em paralelo, o número de assinaturas a determinar e o número total de píxeis e bandas a processar. O desempenho teórico da arquitetura é analisado com base em várias configurações, onde se define o número de bandas e píxeis a processar em paralelo. Conclui-se que é mais vantajoso paralelizar a arquitetura em termos de número bandas do que píxeis.

A arquitetura é implementada para o sensor AVIRIS, que captura 512 píxeis com 224 bandas espetrais em 8,3 ms. Esta processa 614 píxeis com 224 bandas para determinar oito assinaturas espetrais. São implementadas três arquiteturas com paralelismo, onde cada uma processa em paralelo oito, quatro e duas bandas espetrais, e uma outra sem paralelismo.

A frequência máxima de cada arquitetura é de 90, 100, 100 e 120 MHz e necessitam de 1,57, 2,81, 5,60 e 9,31 ms para executar o método. No caso do sensor AVIRIS todas as arquiteturas (expecto a que necessita de 9,31 ms) são adequadas para processamento em tempo-real abordo.

6.2 Trabalho futuro

Num trabalho futuro seria interessante criar um sistema completo de aquisição e análise de dados hiperespetrais. Este sistema consistiria em integrar uma câmera hiperespetral, de baixo custo, à arquitetura implementada.

Integrar à arquitetura um controlador de acesso direto à memória externa da placa, e utilizar essa memória para armazenar os dados a serem processados pela arquitetura, de modo a aumentar assim o número de píxeis a serem processados.

Analisar o desempenho do método quanto à precisão mínima necessária em vír-

gula flutuante personalizada, de modo a definir o número mínimo de bits do expoente e da mantissa.

Referências

[1] Q. Tong, Y. Xue, and L. Zhang, “Progress in Hyperspectral Remote Sensing Science and Technology in China Over the Past Three Decades,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 7, no. 1, pp. 70–91, 2014.

[2] S. Serpico, G. Moser, and A. Cattoni, “Feature reduction for classification purpose,” in *Hyperspectral Data Exploitation: Theory and Applications*, pp. 245–274, Wiley-Interscience, 2007.

[3] G. Vane, R. O. Green, T. G. Chrien, H. T. Enmark, E. G. Hansen, and W. M. Porter, “The airborne visible/infrared imaging spectrometer (AVIRIS),” *Remote Sensing of Environment*, vol. 44, no. 2-3, pp. 127–143, 1993.

[4] L. J. Rickard, R. W. Basedow, E. F. Zalewski, P. R. Silverglate, and M. Landers, “Hydice: an airborne system for hyperspectral imaging,” *Proc. SPIE*, vol. 1937, pp. 173–179, 1993.

[5] T. Cocks, R. Jenssen, A. Stewart, I. Wilson, and T. Shields, “The HyMap Airborne Hyperspectral Sensor: The System, Calibration and Performance.” 1st EARSEL Workshop on Imaging Spectroscopy, pp. 1-7, 1998.

[6] M. E. Schaepman, L. de Vos, and K. I. Itten, “Apex-airborne prism experiment: hyperspectral radiometric performance analysis for the simulation of the future esa land surface processes earth explorer mission,” *Proc. SPIE*, vol. 3438, pp. 253–262, 1998.

[7] J. Pearlman, P. Barry, C. Segal, J. Shepanski, D. Beiso, and S. Carman, “Hyperion, a space-based imaging spectrometer,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 41, no. 6, pp. 1160–1173, 2003.

[8] A. Kumar and A. Chowdhury, “Hyper-spectral imager in visible and near-infrared band for lunar compositional mapping,” *Journal of Earth System Science*, vol. 114, no. 6, pp. 721–724, 2005.

[9] M. D. Lewis, R. W. Gould, R. Arnone, P. Lyon, P. M. Martinolich, R. Vaughan, A. Lawson, T. Scardino, W. Hou, W. Snyder, R. Lucke, M. Corson, M. Montes, and C. Davis, “The hyperspectral imager for the coastal ocean (hico): Sensor and data processing overview,” in *OCEANS, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, pp. 1–9, 2009.

[10] S. J. Hook, “NASA 2014 The Hyperspectral Infrared Imager (HyspIRI) – Science Impact of Deploying Instruments on Separate Platforms,” Jet Propulsion Laboratory, 2014.

[11] S. Hook, “The hyperspectral thermal emission spectrometer (hytes)-early results,” in *IEEE International Geoscience and Remote Sensing Symposium*, 2013.

[12] P. Barry, “Eo-1/hyperion science data user’s guide, level 1_b,” *TRW Space, Defense & Information Systems, Redondo Beach, CA, Rep. HYP. TO*, vol. 1, 2001.

[13] J. M. Bioucas-Dias, A. Plaza, G. Camps-Valls, P. Scheunders, N. Nasrabadi, and J. Chanussot, “Hyperspectral Remote Sensing Data Analysis and Future Challenges,” *IEEE Geoscience and Remote Sensing Magazine*, vol. 1, no. 2, pp. 6–36, 2013.

[14] J. Setoain, C. Tenllado, M. Prieto, D. Valencia, A. Plaza, and J. Plaza, “Parallel hyperspectral image processing on commodity graphics hardware,” in *Parallel Processing Workshops. ICPP Workshops. International Conference on*, pp. 465–472, 2006.

[15] W. Carter, K. Duong, R. Freeman, H. Hsieh, J. Ja, J. Mahoney, L. Ngo, and S. Sze, “A user programmable reconfigurable logic array,” *Proceedings of the Custom Integrated Circuits Conference*, pp. 233–235, 1986.

[16] S. Hauck, “The roles of fpgas in reprogrammable systems,” *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[17] V. Tanaya and W. Xiaofeng, “On-board partial run-time reconfiguration for pico-satellite constellations,” in *Proc. of First NASA/ESA Conference on Adaptive Hardware and System*, pp. 262–269, 2006.

[18] A. Plaza, Q. Du, Y.-L. Chang, and R. King, “High performance computing for hyperspectral remote sensing,” *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 4, no. 3, pp. 528–544, 2011.

[19] S. Lopez, P. Horstrand, G. M. Callico, J. F. Lopez, and R. Sarmiento, “A novel architecture for hyperspectral endmember extraction by means of the modified vertex component analysis (mvca) algorithm,” *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 5, no. 6, pp. 1837–1848, 2012.

[20] J. Rosário, J. M. P. Nascimento, and M. Véstias, “Fpga-based architecture for hyperspectral endmember extraction,” *Proc. SPIE*, vol. 9247, pp. 924703–924703–11, 2014.

[21] D. Valencia, A. Plaza, M. A. Vega-Rodríguez, and R. M. Pérez, “Fpga design and implementation of a fast pixel purity index algorithm for endmember extraction in hyperspectral imagery,” *Proc. SPIE*, vol. 5995, pp. 599508–599508–10, 2005.

[22] C. González, D. Mozos, J. Resano, and A. Plaza, “Fpga for computing the pixel purity index algorithm on hyperspectral images.,” in *International Conference on Engineering of Reconfigurable Systems and Algorithms, ERSA*, p. 125–131, 2010.

[23] C. González, J. Resano, D. Mozos, A. Plaza, and D. Valencia, “Fpga implementation of the pixel purity index algorithm for remotely sensed hyperspectral image analysis,” *EURASIP Journal on Advances in Signal Processing*, vol. 68, pp. 1–13, 2010.

[24] C. González, D. Mozos, J. Resano, and A. Plaza, “Fpga implementation of the n-findr algorithm for remotely sensed hyperspectral image analysis,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 50, no. 2, pp. 374–388, 2012.

[25] C.-I. Chang, W. Xiong, and C.-C. Wu, “Field-programmable gate array design of implementing simplex growing algorithm for hyperspectral endmember extraction,” *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 51, no. 3, pp. 1693–1700, 2013.

[26] C. González, J. Resano, A. Plaza, and D. Mozos, “Fpga implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm,” *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 5, no. 1, pp. 248–261, 2012.

[27] R. Nekovei and M. Ashtijou, “Reconfigurable acceleration for hyperspectral target detection,” in *IEEE International Geoscience and Remote Sensing Symposium.*, pp. 3229–3232, IEEE, 2007.

[28] H. M. Hussain, K. Benkrid, A. T. Erdogan, and H. Seker, “Highly parameterized k-means clustering on fpgas: Comparative results with gpps and gpus,” in *Reconfigurable Computing and FPGAs (ReConFig)*, pp. 475–480, 2011.

[29] Y.-T. Hwang, C.-C. Lin, and R.-T. Hung, “Lossless hyperspectral image compression system-based on hw/sw codesign,” *Embedded Systems Letters, IEEE*, vol. 3, no. 1, pp. 20–23, 2011.

[30] Digilent ZedBoard Zynq, FPGA, “Dev. board documentation,” 2012.

[31] M. Ciznicki, K. Kurowski, and A. Plaza, “Graphics processing unit implementation of jpeg2000 for hyperspectral image compression,” *Journal of Applied Remote Sensing*, vol. 6, no. 1, pp. 061507–1–061507–14, 2012.

[32] N. Keshava and J. Mustard, “Spectral unmixing,” *Signal Processing Magazine, IEEE*, vol. 19, no. 1, pp. 44–57, 2002.

[33] J. B. Adams, M. O. Smith, and P. E. Johnson, “Spectral mixture modeling: A new analysis of rock and soil types at the viking lander 1 site,” *Journal of Geophysical Research: Solid Earth*, vol. 91, no. B8, pp. 8098–8112, 1986.

[34] D. Zhang, J. Yin, and H. Li, “Mapping of surface sediment types in intertidal flat using linear spectral unmixing of hyperion data,” in *Remote Sensing, Environment and Transportation Engineering (RSETE)*, pp. 450–453, 2011.

[35] C.-I. Chang, *Hyperspectral Data Processing: Algorithm Design and Analysis*. John Wiley & Sons, 2013.

[36] D. Gu, A. Gillespie, A. Kahle, and F. Palluconi, "Autonomous atmospheric compensation (aac) of high resolution hyperspectral thermal infrared remote-sensing imagery," *Geoscience and Remote Sensing*, vol. 38, no. 6, pp. 2557–2570, 2000.

[37] I. Olthof, C. Butson, and R. Fraser, "Signature extension through space for northern landcover classification: A comparison of radiometric correction methods," *Remote Sensing of Environment*, vol. 95, no. 3, pp. 290–302, 2005.

[38] W. Moses, *Atmospheric Correction of hyperspectral data: execution of and comparison between FLAASH and TAFKAA_6S*. Cornell University, 2005.

[39] J. Bioucas-Dias and J. Nascimento, "Hyperspectral subspace identification," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 46, no. 8, pp. 2435–2445, 2008.

[40] J. M. Nascimento and J. M. Bioucas Dias, "Vertex component analysis: A fast algorithm to unmix hyperspectral data," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 43, no. 4, pp. 898–910, 2005.

[41] J. W. Boardman *et al.*, "Automating spectral unmixing of aviris data using convex geometry concepts," in *Summaries 4th Annu. JPL Airborne Geoscience Workshop*, vol. 1, pp. 11–14, JPL Publication 93–26, 1993.

[42] M. E. Winter, "N-findr: an algorithm for fast autonomous spectral end-member determination in hyperspectral data," *Proc. SPIE*, vol. 3753, pp. 266–275, 1999.

[43] C.-I. Chang, C.-C. Wu, W.-m. Liu, and Y.-C. Ouyang, "A new growing method for simplex-based endmember extraction algorithm," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 44, no. 10, pp. 2804–2819, 2006.

[44] B. N. Datta, *Numerical linear algebra and applications*. Siam, 2010.

[45] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50. Siam, 1997.

[46] N. J. Higham, *Accuracy and stability of numerical algorithms*. Siam, 2002.

[47] R. N. Clark, G. A. Swayze, R. Wise, K. E. Livo, T. M. Hoefen, R. F. Kokaly, and S. J. Sutley, *USGS Digital Spectral Library splib05a, USGS Open File Report 03-395*. US Geological Survey Reston, VA, 2003.

[48] A. Gelman, J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin, *Bayesian data analysis*. CRC press, 2013.

[49] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[50] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Courier Dover Publications, 1998.

[51] F. Kruse, A. Lefkoff, J. Boardman, K. Heidebrecht, A. Shapiro, P. Barloon, and A. Goetz, "The spectral image processing system (sips)—interactive visualization and analysis of imaging spectrometer data," *Remote sensing of environment*, vol. 44, no. 2, pp. 145–163, 1993.

[52] J. C. Price, "How unique are spectral signatures?," *Remote Sensing of Environment*, vol. 49, no. 3, pp. 181–186, 1994.

[53] J. W. Boardman, F. A. Kruse, and R. O. Green, "Mapping target signatures via partial unmixing of aviris data," in *Summaries 5th JPL Airborne Earth Science Workshop*, vol. 1, pp. 23–26, 1995.

[54] C.-I. Chang, "Spectral information divergence for hyperspectral image analysis," in *IEEE International Geoscience and Remote Sensing Symposium.*, vol. 1, pp. 509–511, 1999.

[55] S. M. Schweizer and J. M. Moura, "Efficient detection in hyperspectral imagery," *Image Processing, IEEE Transactions on*, vol. 10, no. 4, pp. 584–597, 2001.

[56] G. Golub and C. Van Loan, *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press, 2013.

[57] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[58] Xilinx, Inc., “LogiCORE IP Floating-Point Operator v6.1.” Xilinx, Inc., Natick, San Jose, United States, 2012.

[59] Xilinx, Inc., “Logicore ip block memory generator v7.3.” Xilinx, Inc., Natick, San Jose, United States, 2012.

[60] J. E. Sánchez, E. Auge, J. Santalo, I. Blanes, J. Serra-Sagrista, and A. Kiely, “Review and implementation of the emerging ccsds recommended standard for multispectral and hyperspectral lossless image coding,” in *Data Compression, Communications and Processing (CCP)*, pp. 222–228, 2011.

A | Artigo publicado

FPGA-based Architecture for Hyperspectral Endmember Extraction

João Rosário[a], José M. P. Nascimento[a,b], Mário Véstias[a,c]

[a]Instituto Superior de Engenharia de Lisboa, Lisbon, Portugal
[b]Instituto de Telecomunicações, Lisbon, Portugal
[c]INESC-ID , Lisbon, Portugal

ABSTRACT

Hyperspectral instruments have been incorporated in satellite missions, providing data of high spectral resolution of the Earth. This data can be used in remote sensing applications, such as, target detection, hazard prevention, and monitoring oil spills, among others. In most of these applications, one of the requirements of paramount importance is the ability to give real-time or near real-time response. Recently, onboard processing systems have emerged, in order to overcome the huge amount of data to transfer from the satellite to the ground station, and thus, avoiding delays between hyperspectral image acquisition and its interpretation. For this purpose, compact reconfigurable hardware modules, such as field programmable gate arrays (FPGAs) are widely used.

This paper proposes a parallel FPGA-based architecture for endmember's signature extraction. This method based on the Vertex Component Analysis (VCA) has several advantages, namely it is unsupervised, fully automatic, and it works without dimensionality reduction (DR) pre-processing step. The architecture has been designed for a low cost Xilinx Zynq board with a Zynq-7020 SoC FPGA based on the Artix-7 FPGA programmable logic and tested using real hyperspectral data sets collected by the NASA's Airborne Visible Infra-Red Imaging Spectrometer (AVIRIS) over the Cuprite mining district in Nevada. Experimental results indicate that the proposed implementation can achieve real-time processing, while maintaining the methods accuracy, which indicate the potential of the proposed platform to implement high-performance, low cost embedded systems, opening new perspectives for onboard hyperspectral image processing.

Keywords: Hyperspectral Endmember Extraction, Vertex Component Analysis (VCA), Field-Programmable Gate Arrays (FPGA), Onboard Processing.

1. INTRODUCTION

Hyperspectral imagery is a continuously growing area in remote sensing applications. The spectral range extending from the visible region through the near-infrared and mid-infrared in hundreds of narrow contiguous bands, provides a very high spectral resolution, which allows the detection and the discrimination between different chemical elements of the observed image.[1–3] These instruments, because of their potential in remote sensing applications, such as, target detection for security/militar purposes, hazard prevention, and monitoring oil spills among others, have been incorporated in satellite missions.[4] In most of those applications, one of the requirements of paramount importance is the ability to give real-time or near real-time response.[5]

Although, these sensors allow to capture large three dimensional data cubes, for instance Hyperion sensor produces a data cube $256 \times 6925 \times 242$ in 30 seconds,[6] the available downlink bandwidth to ground stations is limited, which brings prohibitive delays that endanger the real-time or near real-time requirements of such applications. Recently, onboard processing systems have emerged, in order to overcome these delays between hyperspectral image acquisition and its interpretation. Such systems need to have high computational performance, compact size, reduced weight and low power consumption among other characteristics. Additionally, it

Further author information: (Send correspondence to J. Nascimento) E-mail: zen@isel.pt

High-Performance Computing in Remote Sensing IV, edited by Bormin Huang, Sebastian López, Zhensen Wu,
Proc. of SPIE Vol. 9247, 924703 · © 2014 SPIE · CCC code: 0277-786X/14/$18 · doi: 10.1117/12.2067039

is desirable that those onboard systems could also be flexible in order to be adaptable to the needs of different missions. Field-programmable gate arrays (FPGA) represent a good choice to achieve the above mentioned requirements.[7,8] In recent years, scientific community have proposed several FPGA implementations of hyperspectral processing techniques, namely for unmixing,[9–14] endmember extraction,[10,15,16] abundance estimation,[17] target detection,[18] clustering,[19] image compression,[20] among others.

The main problem of hyperspectral imagery is the low spatial resolution due to the instrument instantaneous field of view (IFOV), thus, the pixel ground size can vary from a few to tens of meters. This means that each pixel is a mixture of several spectrally distinct materials (also called *endmembers*).[1,21] The endmembers are generally assumed to represent the pure materials present in the image. A wide class of methods, called geometrical approaches, aim to infer the set of endmembers, at a low computational complexity. These methods assume the linear mixture model and the presence of pure pixels in the dataset, *i.e.*, pixels containing just one endmember.

This paper proposes a new endmember extraction architecture designed for FPGA. This architecture is based on a geometrical based approach, called Vertex Component Analysis (VCA).[22] Briefly, VCA iteratively projects the pixels onto a direction orthogonal to the subspace spanned by the endmembers already determined. The new endmember signature corresponds to the extreme of the projection. VCA has several advantages, namely it is unsupervised, fully automatic, and it works with or without dimensionality reduction (DR) pre-processing step. This is of paramount importance since the DR step often amounts to compute the eigen-values of the correlation matrix of the dataset, which can take a larger time consumption not compatible with real-time requirements. Additionally, in order to avoid the computation of an inverse matrix typically performed by a singular value decomposition (SVD) present in the seminal VCA work[22] herein a QR decomposition is conducted to reuse the orthogonal vectors already determined in previous iterations.[9] Since the projection is performed by the multiply-accumulate operation for each band, which does not depend on the others bands operation nor on the others pixels projection, the projection is designed to be implemented in parallel on each iteration, reducing the processing time.

One of this work's primary goal is the methods design for a low cost embedded system for onboard use. The system does not have hardware resources to compute all pixel projection at once, so the extraction of the endmembers is conducted for each line of the data cube as they are acquired by the sensor. At the end a final selection is done to select the set of endmembers.

In order to validate the implementation, a quantitative study is developed using a Xilinx Zynq board with a Zynq-7020 SoC FPGA based on the Artix-7 FPGA programmable logic. The method is tested using real hyperspectral datasets publicly available. The experimental results indicate that the proposed implementation can achieve real-time performance in the considered FPGA, while maintaining the methods accuracy, thus being able to extract the endmembers signatures in real-time, which makes our reconfigurable system appealing for on-board hyperspectral data processing.

The remainder of this paper is organized as follows. Section 2 formulates the problem and briefly describes the VCA method. Section 3 describes the method design and implementation for the FPGA. Section 4 provides the experimental accuracy and performance results of the architecture implementation on an FPGA using the well-known Cuprite hyperspectral dataset. Section 5 concludes the paper with some remarks and draws some future research work line.

2. ALGORITHM

Linear mixing model considers that a mixed pixel is a linear combination of endmember signatures weighted by the correspondent abundance fractions, *i.e.* at each pixel abundances represent the percentage of each endmember that is present in the pixel. Assuming that $\mathbf{Y} \equiv [\mathbf{y}_1, \ldots, \mathbf{y}_N] \in \mathbb{R}^{L \times N}$ denotes a matrix holding the N observed spectral vectors, this matrix is given by

$$\mathbf{Y} = \mathbf{MS} + \mathbf{W}, \tag{1}$$

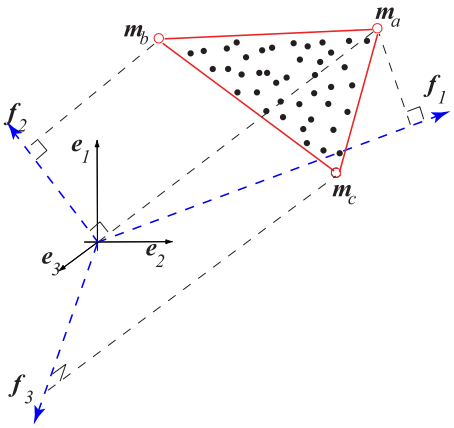


Figure 1. Hyperspectral mixture of three endmembers illustrating the VCA algorithm.

where $\mathbf{M} \equiv [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_p]$ is a full-rank $L \times p$ mixing matrix ($\mathbf{m}_j$ denotes the jth endmember signature), p is the number of endmembers present in the covered area, $\mathbf{S} \equiv [\mathbf{s}_1, \ldots, \mathbf{s}_N] \in \mathbb{R}^{p \times N}$ is a matrix holding the respective abundance fractions, and $\mathbf{W} \equiv [\mathbf{w}_1, \ldots, \mathbf{w}_N] \in \mathbb{R}^{L \times N}$ accounts for additive noise.

To be physically meaningful,[1,21] abundance fractions are subject to nonnegativity and full additivity constraints, *i.e.*, abundance fractions are in the $p-1$ probability simplex:

$$\{\mathbf{S} \in \mathbb{R}^{p \times N} : \mathbf{S} \succeq 0\,,\, \mathbf{1}_p^T \mathbf{S} = \mathbf{1}_n^T\}, \tag{2}$$

where $\mathbf{S} \succeq 0$ means $s_{ij} \geq 0$, for $i = 1, \ldots, p$, and $j = 1, \ldots, N$, $\mathbf{1}_p$ and $\mathbf{1}_N$ denote a $1 \times p$ and $1 \times N$ column vectors filled of 1's. Thus, under the linear mixing model the observed spectral vectors in a given scene are in a simplex whose vertices correspond to the endmembers. Assuming that each endmember has at least one pure pixel on the dataset, endmember extraction amounts to find those pure pixels.

VCA is one the most popular endmember extraction algorithms within the literature. VCA is an unsupervised method to extract endmembers from hyperspectral datasets and is based on the geometry of convex sets. It exploits two facts: 1) the endmembers are the vertices of a simplex and 2) the affine transformation of a simplex is also a simplex. VCA is a fully automatic algorithm and it works with or without DR pre-processing step. The algorithm iteratively projects data onto a direction orthogonal to the subspace spanned by the endmembers already determined. The new endmember signature corresponds to the pixel with largest projection. The algorithm repeats the procedure until the whole set of p endmembers is found. Figure 1 illustrates the VCA method working on a simplex defined by a mixture of three endmembers. In the first iteration, data is projected onto the first direction $\mathbf{f}_1$. The extreme of the projection corresponds to endmember $\mathbf{m}_a$. In the next iteration, endmember $\mathbf{m}_b$ is found by projecting data onto direction $\mathbf{f}_2$, which is orthogonal to $\mathbf{m}_a$. Finally, a new direction $\mathbf{f}_3$, orthogonal to the subspace spanned by $\mathbf{m}_a$ and $\mathbf{m}_b$ is generated and the endmember $\mathbf{m}_c$ is found by seeking the extreme of the projection of the dataset onto $\mathbf{f}_3$.

Algorithm 1, shows the main steps of the VCA, without the DR step. Although the number of endmembers is much lower than the number of bands ($p \ll L$) and, thus, it is advantageous, in terms of signal-to-noise ratio (SNR) to represent the spectral vectors in a signal subspace basis,[23] this step, as referred before, can take a larger time consumption, which could not be compatible with real-time requirements. More details on the full VCA algorithm can be found in [22]. In Algorithm 1 symbol $\mathbf{A}$ denotes the estimated mixing matrix and $[\mathbf{A}]_{:,j}$ stands for jth column of $\mathbf{A}$.

Algorithm 1 : VCA

1: $\mathbf{A} := [\mathbf{e}_u \,|\, \mathbf{0} \,|\, \ldots \,|\, \mathbf{0}]$; $\{\mathbf{e}_u := [0, \ldots, 0, 1]^T$ and $\mathbf{A}$ is a $L \times p$ auxiliary matrix$\}$
2: **for** $i := 1$ to p **do**
3: $\quad \mathbf{w} := \text{randn}\,(0, \mathbf{I}_p)$; { $\mathbf{w}$ is a zero-mean random Gaussian vector}
4: $\quad \mathbf{f} := (\mathbf{I} - \mathbf{A}\mathbf{A}^{\#})\mathbf{w}$; {$\mathbf{f}$ is a vector orthogonal to the subspace spanned by $\mathbf{A}_{:,1:i}$}
5: $\quad \mathbf{v} := \mathbf{f}^T\mathbf{Y}$;
6: $\quad k := \arg\max_{j=1,\ldots,N} |[\mathbf{v}]_{:,j}|$;
7: $\quad [\mathbf{A}]_{:,i} := [\mathbf{Y}]_{:,k}$;
8: **end for**

3. FPGA ARCHITECTURE DESIGN AND IMPLEMENTATION

The methodology followed to design the hardware architecture starts with an analysis of the algorithm to identify the most computationally demanding steps followed by a detailed description of how each step of the algorithm is calculated. Then, a hardware architecture that supports the execution of each step of the algorithm is proposed.

3.1 Analysis and Implementation of the VCA Method

Knowing that the number of endmembers is much lower than the number of bands ($p \ll L$) and that it is advantageous in terms of SNR to represent the spectral vectors in a lower subspace,[23] previous hardware approaches assume there exists a pre-processing (DR) step to reduce the dimension of the data space before entering the hardware system, and consequently to simplify the hardware. However, the time taken by the DR step is quite large and is not compatible with real-time requirements making these hardware solutions not applicable to real-time processing. Since VCA also works without this step, the proposed hardware implementation does not consider a DR pre-processing step.

The most complex task of VCA is the calculation of vector $\mathbf{f}$ (see line 4 on Algorithm 1), which requires the computation of a pseudo-inverse matrix, typically using the SVD,[24] which is numerically very stable and accurate,[25] but it is computationally very complex and demanding. The projection of the image onto the orthogonal direction (see line 5 on Algorithm 1) is also computationally very demanding, due to the large spatial dimensions of the image. To reduce the complexity and improve the performance of the hardware implementation, several optimizations have been considered in the implementation of the VCA algorithm (see Algorithm 2).

In Algorithm 2 a QR decomposition using the classic Gram-Schmidt process[24] is used to determine vector $\mathbf{f}$ on Algorithm 1 instead of the SVD method. The QR decomposition is computationally less complex and demanding than SVD, however it is numerically less stable and accurate.[26] To reduce the computation time of the QR decomposition an incremental procedure is used. The procedure consists on reusing the orthogonal vectors Q determined so far, thus only one orthogonal vector needs to be computed on each iteration (see lines 9–17 on algorithm 2). Moreover, instead of normalizing the vector by performing multiple divisions (lines 16 and 17 on algorithm 2), the reciprocal of the norm of the vector is calculated first and then multiplied by the vector. Since the multiplication operation is faster and uses less hardware resources compared to the division operation, a parallel implementation of the operation becomes faster and smaller when implemented in hardware.[27,28]

The projection of the image onto a direction to obtain vector $\mathbf{v}$ on Algorithm 1 consists on a vector-matrix multiplication (see lines 22–28 on Algorithm 2). The operation was speeded up by parallelizing the dot-product operation (sub–routine *Dot_product* in line 23 on algorithm 2) in order to process several bands on each cycle reducing proportionally the number of cycles needed to project a single pixel. The determination of $\mathbf{k}$ on Algorithm 1, the maximum entry of vector $\mathbf{v}$, is done in parallel with the dot-product operations, that is, the result of each dot-product is immediately compared with the previous entry to find and keep the maximum value. This optimization avoids the storage of the result of the dot product and reduces the execution time of the algorithm.

Algorithm 2 Vertex Component Analysis (VCA) for hardware implementation

INPUT Y
$\mathbf{w} := rand(1, L)$;
$\mathbf{temp} := \mathbf{e}_u$;
$\mathbf{f} := \mathbf{w}$;
$j := 1$;
for $i := 1$ to 8 **do**
  if $i > 1$ **then**
    $\mathbf{temp} := \mathbf{Y}(:, \mathbf{index}(i-1))$;
    if $i > 2$ **then**
      $j := j + 1$;
      for $k := 1$ to $j - 1$ **do**
        $r := Dot\_product(\mathbf{Q}(:, k), \mathbf{temp})$;
        $\mathbf{temp} := \mathbf{temp} - r \times \mathbf{Q}(:, k)$;
      end for
    end if
    $r := 1/\sqrt{Dot\_product(\mathbf{temp}, \mathbf{temp})}$;
    $\mathbf{Q}(:, j) := r \times \mathbf{temp}$;
    $proj := Dot\_product(\mathbf{w}, \mathbf{Q}(:, j))$;
    $\mathbf{f} := \mathbf{f} - proj \times \mathbf{Q}(:, j)$;
  end if
  $v\_old = 0$;
  for $k := 1$ to 614 **do**
    $v := Dot\_product(\mathbf{f}, \mathbf{Y}(:, k))$;
    if $v > v\_old$ **then**
      $v\_old := v$;
      $\mathbf{index}(i) := k$;
    end if
  end for
end for
OUPUT index

3.2 FPGA Architecture

This section presents the proposed hardware implementation of Algorithm 2 described in the previous section. The architecture is designed to support real-time processing of hyperspectral images acquired from AVIRIS sensor.

The architecture consists of a general-purpose processor (*ARM*) used to send the image to the hardware and to get the index of the pixels that were identified as endmembers by the hardware, a random number generator (*Random-generator*), a memory module (*Memory*) to store inputs and outputs of the algorithm, as well as temporary vectors and variables, an integer to float converter (*Int2float*), a dot-product calculator (*Dot-product*), a multiply-subtract module (*Multiplier-Subtrator*), a reciprocal square-root calculator used to normalize the last coefficient r of the QR decomposition (*Reciprocal-square-root*), and a floating-point comparator (*Comparator*) used to determine the biggest argument of all projections (see Figure 2).

Following the algorithm 2 described previously, the ARM starts transferring the hyperspectral image to the *Memory* module and then sends a control signal to the dedicated hardware. Initially, a pseudo-random vector ($\mathbf{w}$) is generated using the *Random-generator* module. At the same time a QR decomposition is computed using

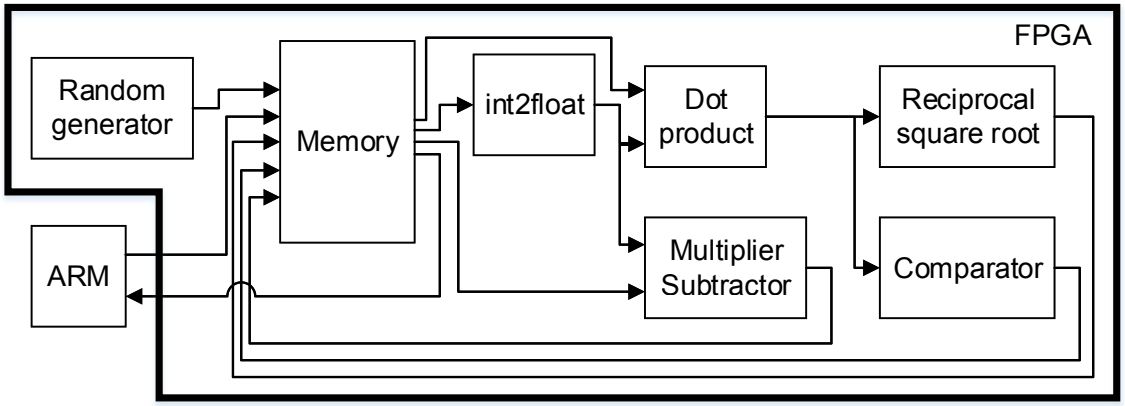


Figure 2. General overview of the VCA hardware architecture

modules *Dot-product*, *Multiplier-Subtractor*, and *Reciprocal-square-root* that generate a new orthogonal vector. The result of this decomposition and its intermediate results are stored in *Memory*. Then, the orthogonal direction $\mathbf{f}$ is computed, using modules *Dot-product* and *Multiplier-Subtractor* and stored in *Memory*. Finally, the projection of image $\mathbf{Y}$ on vector $\mathbf{f}$ is computed using module *Dot-product*. As the results are produced they are sent to module *Comparator* that determines the index of the highest projection value. At the end of all processing a control signal is sent to the processor indicating the end of the algorithm and that the indexes of those pixels found as endmembers are available in memory.

In order to efficiently utilize the available resources, some modules are reused to implement different steps of the algorithm. For example, the *Dot-product* and the *Multiplier-Subtractor* modules are both used on the QR decomposition and the image projection. Also, all modules operates on single-precision floating-point data.[29] Since the data image is stored as 15-bit integers, the *Int2float* module is used to convert from 15-bit integers to single-precision floating-point to be used on the *Dot-product* and in the *Multiplier-Subtrator* modules.

The *Random-generator* module generates 32-bit integer numbers using the Mersenne twister algorithm.[30] The integers generated are then rearranged to form a single-precision floating point number, avoiding the use of an integer to floating point converter.

The *Memory* module is composed by eight simple dual port random access memories (RAM) with different datawidths. One RAM has a datawidth of 10 bits and is used to store the results of the algorithm. Another, is used to store the input image and therefore has a datawidth of 15 bits. Other two with 32 bits of datawidth are used to store the random vector and the orthogonal direction. Three other RAM modules with 1, 31, and 32 bits of datawidth are used by the *Random-generator* module. A last single port RAM of 32 bits is used to store the orthogonal vectors of the current base, and also to implement a FIFO to store the coefficients R produced by the QR decomposition and two registers of 32 bits to store the last coefficient from QR and intermediate results.

The *Multiplier-Subtractor* module is used to calculate a multiply-subtract operation over vectors. The module is designed also with eight multipliers and subtractors capable to process eight bands at the same time.

The *Dot-product* block is used to multiply vectors and is composed by parallel multiply-add blocks, *DP*, and an adder tree (see Figure 3(left)). Each multiply-add block (see Figure 3 (right)) consists of a multiplier followed by an accumulator. The accumulator is designed with three cycles of latency to allow an higher operating frequency. Every block computes the dot product of n ($= \frac{224}{p}$) bands, where p is the number of parallel multiply-add blocks, which are then added with a tree adder.

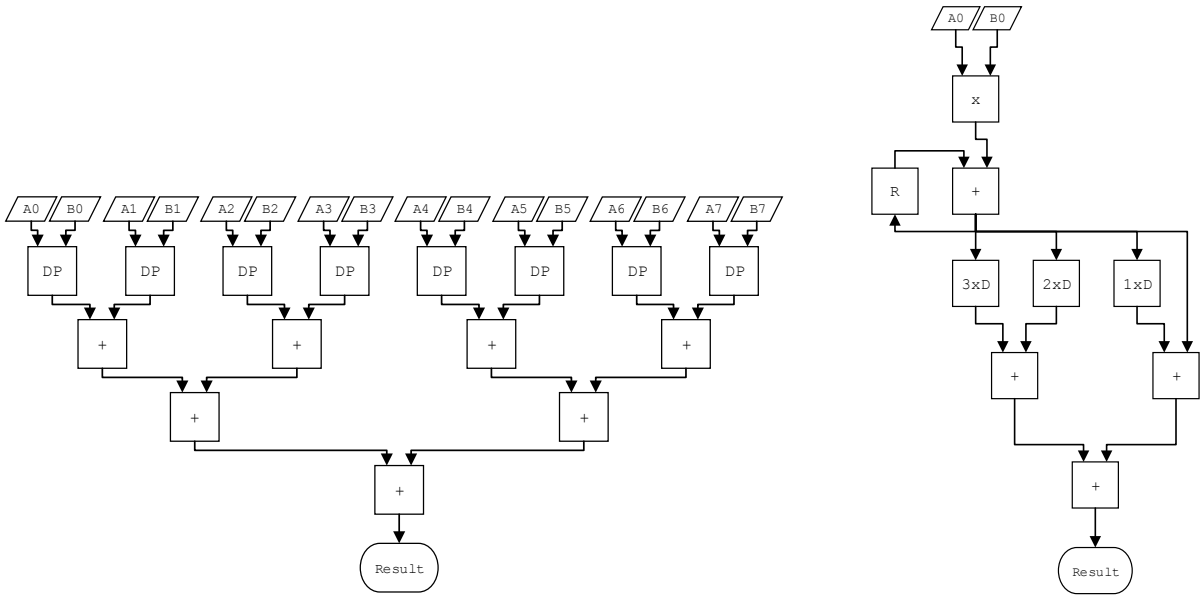


Figure 3. *Dot-product* module architecture: (left) full architecture, (right) detail of DP submodule.

This particular implementation uses eight parallel multiplier blocks capable to process eight spectral bands each clock cycle. Since the method is scalable, more parallel multiplier blocks can be considered, which would reduce the number of computation cycles and consequently the total processing time of the algorithm. The scalability is only limited by the available hardware resources. In the proposed architecture, the number of parallel multipliers was limited to eight due to the available hardware resources of the target device, as will be shown in Section 4.

4. EXPERIMENTAL RESULTS

The proposed hardware architecture has been described in VHDL and implemented on a Xilinx Zynq Zedboard with a XC7Z020 SoC device. The hardware design has been done with Xilinx ISE, PlanAhead, Platform Studio and Software Development Kit tools.

The target FPGA is a system on a chip that contains a Dual ARM® Cortex™-A9, and also a reconfigurable area equivalent to an Artix®-7. This family of FPGAs is quite appropriate to develop embedded systems making these boards ideal for fast prototyping and proof-of-concept development.

All basic modules, including multipliers, adders, subtracters and memories were generated using the Xilinx Core generator tool. The floating-point units were generated with the Xilinx LogiCORE IP CORE Floating point v6.1,[31] and memories were generated using the Xilinx LogiCORE IP Block Memory Generator v7.3.[32]

The following sections describe the endmembers extraction accuracy results performed by the developed hardware architecture and the implementation results in terms of used hardware resources and working frequency.

4.1 Endmember Extraction Accuracy Evaluation

The hardware implementation was evaluated with real hyperspectral data collected by AVIRIS[33] sensor over Cuprite mining district at Nevada. The Cuprite site has been extensively used for remote sensing experiments over the past years, is well understood mineralogically, and has several exposed minerals of interest, all included in the USGS library considered in experiments. The dataset is a subimage of 250x191 pixels and 224 spectral

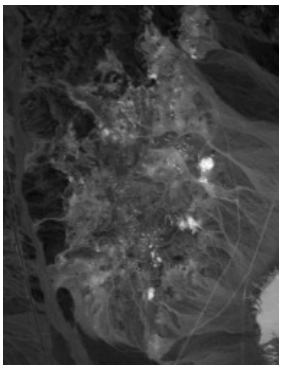


Figure 4. Band 30 (wavelength $\lambda = 647.7nm$) of the subimage of AVIRIS Cuprite Nevada dataset.

bands between 0.4 and 2.5 μm, with a spectral resolution of 10 nm of the original dataset. This subimage is already in reflectance units and the bands of water absorption (104-113 and 148-167) were removed, along with the low SNR bands (1-2 and 221-224), leaving a total of 188 spectral bands in the experiment. Fig. 4 shows band 30 (wavelength $\lambda = 647.7nm$) of the subset considered.

In order to process all 47750 pixels of the dataset, the architecture processes blocks of 614 pixels each time. The endmembers found on the last iteration are grouped with the next block of pixels.

To measure the accuracy of the implemented hardware the well known spectral angle distance (SAD) is adopted. Table 1 presents the SAD between the extracted endmembers and the closest laboratory reflectance for the substances that are most representative on the scene. For accuracy comparison purposes it is also shown the results for the same dataset with DR on the third column. It is worth noting that the displayed results show a similar accuracy with the ones presented in [22]. Andradite, Buddingtonite, and Kaolinite are the substances where the results presented is slightly worse.

Substance	**SAD** without DR	**SAD** with DR
Alunite	3.95	3.78
Andradite	4.36	3.27
Buddingtonite	4.63	3.99
Kaolinite	4.92	4.96
Montmorillonite	2.87	1.51
Muscovite	3.78	3.44
Nontronite	3.18	2.63
Pyrope	2.22	1.71

Table 1. SAD results between extracted endmembers and laboratory reflectances for hardware VCA.

4.2 Implementation results

Due to the limited hardware resources the architecture is capable to extract only eight endmembers signatures from a dataset composed by 614 pixels with 224 spectral bands, which corresponds to one line of an image acquired by the AVIRIS sensor (it can also process spatial and/or spectral smaller datasets by simply filling the unused pixels with zeros). For larger datasets, after the endmembers's signatures extraction of each line, a final selection is done to select the set of endmembers.

Table 2 summarizes the resources used to implement the architecture. The maximum frequency achieved

Resource	Utilization
Register	18712 (18%)
LUT	20102 (38%)
Slice	7720 (58%)
RAM36Kb	79 (56%)
DSP48	119 (55%)

Table 2. Summary of the hardware resources used for the FPGA implementation of the VCA algorithm.

is 90 MHz. It is noteworthy that the designed hardware assumed that all bands are available, *i.e.*, the image does not have any pre-processing to reduce its spectral dimensionality, so the size of the image is much bigger in comparison with the subspace projected image. It occupies 56% of all RAM36Kb available on the FPGA, creating demanding physical routing constraints to connect all these RAMs on the FPGA.

To extract eight endmembers from an image of 614 pixels, the architecture needs 1.6 ms, which corresponds to 144000 clock cycles (137600 cycles to do the image projections and the remaining 6400 cycles to compute the orthogonal directions), assuming that all the data is already in memory. Since the AVIRIS sensor acquires 512 pixels of 224 spectral bands in 8.3 ms,[34] the implemented architecture is appropriate for real-time on-board hyperspectral data processing.

Table 3 presents a comparison between the Modified VCA (MVCA)[9] architecture for eight endmembers and the proposed scheme.

Resource	VCA	MVCA
Register	18712	24000
LUT	20102	18500

Table 3. Summary of resource utilization for the FPGA implementation of the VCA algorithm and MVCA algorithm.

The MVCA architecture does not use any embedded DSP blocks, which means that every floating-point operation is implemented only with registers and LUTs. Also, the MVCA architecture parallelizes all modules for the total number of bands since it was assumed that the hyperspectral image has been DR pre-processed before running the algorithm.

The proposed architecture is scalable and so better performance results could be achieved using an FPGA with more resources.

5. CONCLUSIONS

Onboard processing systems have recently emerged, in order to overcome the huge amount of data to transfer from the satellite to the ground station. Hyperspectral imagery is a remote sensing technology that can benefit of onboard processing. This paper proposes an FPGA-based architecture for endmember's signature extraction. This method based on the Vertex Component Analysis (VCA) is fully automatic and it works without dimensionality reduction (DR) pre-processing step.

The proposed architecture has been designed for a Xilinx Zynq board with a Zynq-7020 SoC FPGA. Experimental results with real hyperspectral datasets indicate that the proposed implementation can fulfill real-time requirements in a low cost SoC FPGA, while maintaining the methods accuracy. As future research lines, we intent to optimize the design of this method in order to achieve higher working frequencies, opening new perspectives for low cost onboard hyperspectral image processing.

ACKNOWLEDGMENTS

The authors would like to take this opportunity to gratefully thank Fundação para a Ciência e a Tecnologia and Instituto de Telecomunicações support under project PEst-OE/EEI/LA0008/2013.

REFERENCES

[1] Bioucas-Dias, J., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., and Chanussot, J., "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observations Rem. Sens.* **99**(1-16) (2012).

[2] Parente, M. and Plaza, A., "Survey of geometric and statistical unmixing algorithms for hyperspectral images," in [*2nd IEEE GRSS Workshop on Hyperspectral Image and Sig. Proc.-WHISPERS'2010*], (2010).

[3] Bioucas-Dias, J. M. and Plaza, A., "Hyperspectral unmixing: geometrical, statistical, and sparse regression-based approaches," *Image and Sig. Proc. for Rem. Sens. XVI* **7830**(1), SPIE (2010).

[4] Lopez, S., Vladimirova, T., Gonzalez, C., Resano, J., Mozos, D., and Plaza, A., "The promise of reconfigurable computing for hyperspectral imaging onboard systems: A review and trends," *Proceedings of the IEEE* **101**, 698–722 (2013).

[5] Plaza, A. and Chang, C.-I., eds., [*High Performance Computing in Remote Sensing*], Chapman Hall CRC (2007).

[6] "Eo-1/ hyperion science data users guide," tech. rep., TRW Space, Defense & Information Systems (2001).

[7] Plaza, A. and Chang, C.-I., "Clusters Versus FPGA for Parallel Processing of Hyperspectral Imagery," *International Journal of High Performance Computing Applications* **22**(4) (2008).

[8] Plaza, A., Du, Q., Chang, Y.-L., and King, R., "High performance computing for hyperspectral remote sensing," *IEEE J. Sel. Topics Appl. Earth Observations Rem. Sens.* **4**(3), 528 –544 (2011).

[9] Lopez, S., Horstrand, P., Callico, G., Lopez, J., and Sarmiento, R., "A low-computational-complexity algorithm for hyperspectral endmember extraction: Modified vertex component analysis," *IEEE Geosci. Rem. Sens. Let.* **9**(3), 502 –506 (2012).

[10] Gonzalez, C., Mozos, D., Resano, J., and Plaza, A., "FPGA implementation of the n-findr algorithm for remotely sensed hyperspectral image analysis," *Geosci. and Rem. Sens., IEEE Trans. on* **50**, 374–388 (2012).

[11] Nordin, A., Hsu, C. C., and Szu, H. H., "Design of FPGA ica for hyperspectral imaging processing," *proc. of SPIE* **4391**, 444–454 (2001).

[12] Lavenier, D., Theiler, J., Szymanski, J., Gokhale, M., and Frigo, J., "FPGA Implementation of the pixel purity index algorithm," in [*Proc. of the SPIE Photonics East, Workshop on Reconfigurable Architectures*], (2000).

[13] Valencia, D., Plaza, A., Vega-Rodrguez, M. A., and Prez, R. M., "FPGA design and implementation of a fast pixel purity index algorithm for endmember extraction in hyperspectral imagery," *proc. of SPIE* **5995**,(2005).

[14] Hsueh, M. and Chang, C.-I., "Field programmable gate arrays (FPGA) for pixel purity index using blocks of skewers for endmember extraction in hyperspectral imagery," *International Journal of High Performance Computing Applications* **22**(4), 408–423 (2008).

[15] Chang, C.-I, Xiong, W., and Wu, C.-C., "Field-programmable gate array design of implementing simplex growing algorithm for hyperspectral endmember extraction," *Geosci. and Rem. Sens., IEEE Trans. on* **51**, 1693–1700 (2013).

[16] Bernabe, S., Lopez, S., Plaza, A., Sarmiento, R., and Rodriguez, P., "FPGA design of an automatic target generation process for hyperspectral image analysis," in [*Parallel and Distributed Systems (ICPADS), 2011 IEEE 17th International Conference on*], 1010–1015 (2011).

[17] Gonzalez, C., Resano, J., Plaza, A., and Mozos, D., "FPGA implementation of abundance estimation for spectral unmixing of hyperspectral data using the image space reconstruction algorithm," *IEEE J. Sel. Topics Appl. Earth Observations Rem. Sens.* **5**, 248–261 (2012).

[18] Nekovei, R. and Ashtijou, M., "Reconfigurable acceleration for hyperspectral target detection," in [*Geosci. and Rem. Sens. Symposium, 2007. IGARSS 2007. IEEE International*], 3229–3232 (2007).

[19] Hussain, H. M., Benkrid, K., Erdogan, A. T., and Seker, H., "Highly parameterized k-means clustering on fpgas: Comparative results with gpps and gpus," in [*International Conference on Reconfigurable Computing and FPGAs*], 475–480 (2011).

[20] Hwang, Y.-T., Lin, C.-C., and Hung, R.-T., "Lossless hyperspectral image compression system-based on hw/sw codesign," *Embedded Systems Letters, IEEE* **3**, 20–23 (2011).

[21] Nascimento, J. M. P. and Bioucas-Dias, J. M., "Does Independent Component Analysis Play a Role in Unmixing Hyperspectral Data?," *IEEE Trans. Geosci. Rem. Sens.* **43**(1), 175–187 (2005).

[22] Nascimento, J. M. P. and Bioucas-Dias, J. M., "Vertex Component Analysis: A Fast Algorithm to Unmix Hyperspectral Data," *IEEE Trans. Geosci. Rem. Sens.* **43**(4), 898–910 (2005).

[23] Bioucas-Dias, J. M. and Nascimento, J. M. P., "Hyperspectral Subspace Identification," *IEEE Trans. Geosci. Rem. Sens.* **46**(8), 2435–2445 (2008).

[24] Golub, G. H. and Loan, C. F. V., [*Matrix Computations*], Mathematical Sciences, John Hopkins University Press, third ed. (1996).

[25] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., [*Numerical Recipes: The Art of Scientific Computing*], Cambridge University Press, New York, NY, USA, 3rd ed. (2007).

[26] Higham, N. J., [*Accuracy and Stability of Numerical Algorithms*], Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd ed. (2002).

[27] Ercegovac, M., Lang, T., Muller, J.-M., and Tisserand, A., "Reciprocation, square root, inverse square root, and some elementary functions using small multipliers," *IEEE Transactions on Computers* **49**, 628–637 (2000).

[28] Blinn, J., "Floating-point tricks," *Computer Graphics and Applications, IEEE* **17**, 80–84 (1997).

[29] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2008* , 1–70 (2008).

[30] Matsumoto, M. and Nishimura, T., "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Trans. Model. Comput. Simul.* **8**, 3–30 (1998).

[31] Xilinx, "Logicore ip floating-point operator v6.1," (July 2012).

[32] Xilinx, "Logicore ip block memory generator v7.3," (Dec. 2012).

[33] Vane, G., Green, R., Chrien, T., Enmark, H., Hansen, E., and Porter, W., "The Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Rem. Sens. of the Environ.* **44**, 127–143 (1993).

[34] Green, R. O., Eastwood, M. L., Sarture, C. M., Chrien, T. G., Aronsson, M., Chippendale, B. J., Faust, J. A., Pavri, B. E., Chovit, C. J., Solis, M., Olah, M. R., and Williams, O., "Imaging Spectroscopy and the Airborne Visible/Infrared Imaging Spectrometer (AVIRIS)," *Rem. Sens. of the Environ.* **65**(3), 227–248 (1998).