UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Implementación hardware del algoritmo Vertex Component Analysis (VCA) para el procesamiento de imágenes hiperespectrales

Autor: Pablo Horstrand Andaluz
Tutor(es): Dr. D. Roberto Sarmiento Rodríguez
Dr. D. Sebastián López Suárez
Fecha: Julio 2011

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Implementación hardware del algoritmo Vertex Component Analysis (VCA) para el procesamiento de imágenes hiperespectrales

HOJA DE FIRMAS

Alumno/a:	Pablo Horstrand Andaluz	Fdo.:
Tutor/a:	Dr. D. Roberto Sarmiento Rodríguez	Fdo.:
Tutor/a:	Dr. D. Sebastián López Suárez	Fdo.:

Fecha: Julio 2011

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Implementación hardware del algoritmo Vertex Component Analysis (VCA) para el procesamiento de imágenes hiperespectrales

HOJA DE EVALUACIÓN

Calificación:

Presidente	Roberto Sarmiento Rodríguez	Fdo.:
Secretario	Luis Hernández Acosta	Fdo.:
Vocal	Pablo Hernández Morera	Fdo.:

Fecha: Julio 2011

IUMA
t +34 928 451 086
f +34 928 451 083
iuma@iuma.ulpgc.es
www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Índice

Índice de figuras

Índice de tablas

1.Introducción

En este capítulo se realiza una breve introducción a las imágenes hiperespectrales haciendo especial hincapié en la necesidad de contar con técnicas de aceleración en el cómputo de los algoritmos de procesamiento de estas imágenes, tales como la paralelización.

Debido a estas elevadas exigencias elevadas en cuanto al coste computacional de los algoritmos se refiere, se recogen diferentes plataformas hardware que son presentadas en el capítulo con sus ventajas e inconvenientes, destacándose el uso de las FPGAs, que es la tecnología utilizada para mapear el algoritmo implementado en este trabajo.

En el capítulo también se muestra una explicación del fenómeno de mezcla de elementos que sucede en la imagen hiperespectral, que da origen a la utilización del modelo lineal de desmezclado.

Por último se presentan los objetivos generales del trabajo que se aborda, y una descripción general de la organización de la memoria.

1.1. Concepto de imagen hiperespectral

Una imagen hiperespectral es una imagen en la que cada punto no viene descrito por un sólo valor de intensidad (como en una imagen en blanco y negro) o por tres componentes de color (como en una imagen RGB de la pantalla del ordenador), sino por un vector de valores espectrales que se corresponden con la contribución de la luz detectada en ese punto a estrechas bandas del espectro. De esta manera un sensor hiperespectral contiene típicamente un número de bandas que va desde las decenas hasta los varios centenares. En muchas ocasiones, este conjunto de bandas no está limitado estrictamente al visible sino que también abarca el infrarrojo y el ultravioleta.

Generalmente la imagen se representa como un cubo (denominado hipercubo o cubo hiperespectral), compuesto por una pila de imágenes correspondientes a cada uno de los planos 2D para cada una de las longitudes de onda. En la figura 1.1, se observa un ejemplo de cubo hiperespectral del que se han separado las bandas a diferentes longitudes de onda.

Estableciendo una comparación con las imágenes RGB, que son las imágenes más comunes, se puede decir que estas son un caso particular de las imágenes hiperespectrales, ya que cada banda de color, rojo, verde y azul, corresponde a una longitud de onda concreta que se selecciona de la imagen hiperespectral para su composición (en el caso de no encontrarse exactamente la longitud de onda de color se toma el valor más cercano, o se realiza una interpolación) formando lo que se denomina imagen de falso color.

La separación entre longitudes de onda viene determinada por la tecnología del sensor que se emplea. Cuanto menor es la separación entre bandas, es decir, a mayor resolución espectral, se esperan mejores resultados en la aplicación a la par que se

amplía el rango de las mismas. No obstante, la cantidad de información y consecuentemente el coste computacional aumenta considerablemente.

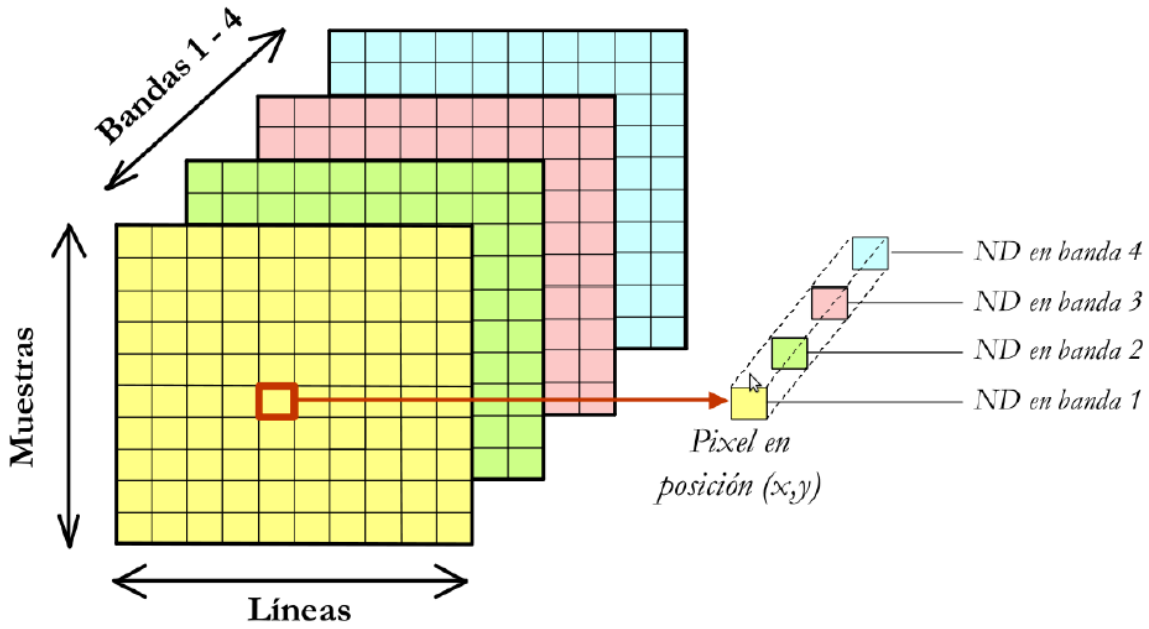


Figura 1.1. Concepto de imagen hiperespectral.

Si la separación entre bandas es muy grande y la imagen dispone de un número pequeño de las mismas (de 4 a 20 aproximadamente), la imagen se denomina multiespectral. A pesar de basarse en un mismo concepto, las aplicaciones en ambos casos son bastante diferentes, y es que en el caso de las imágenes multiespectrales la información proporcionada por un píxel en ocasiones resulta limitada. Por lo tanto los métodos de procesamiento son diferentes entre una tecnología y la otra.

Otro de los aspectos importantes a destacar de este tipo de imágenes, es que en ellas es habitual la existencia de mezclas a nivel de subpíxel, por lo que a grandes rasgos podemos encontrar dos tipos de píxeles en estas imágenes: píxeles puros y píxeles mezcla [1]. Se puede definir un píxel mezcla como aquel en el que cohabitan diferentes materiales. Este tipo de píxel son los que constituyen la mayor parte de la imagen hiperespectral, en parte, debido a que este fenómeno es independiente de la escala considerada ya que tiene lugar incluso a niveles microscópicos [2]. La figura 1.2 muestra

un ejemplo del proceso de adquisición de píxeles puros (a nivel macroscópico) y mezcla en imágenes hiperespectrales.

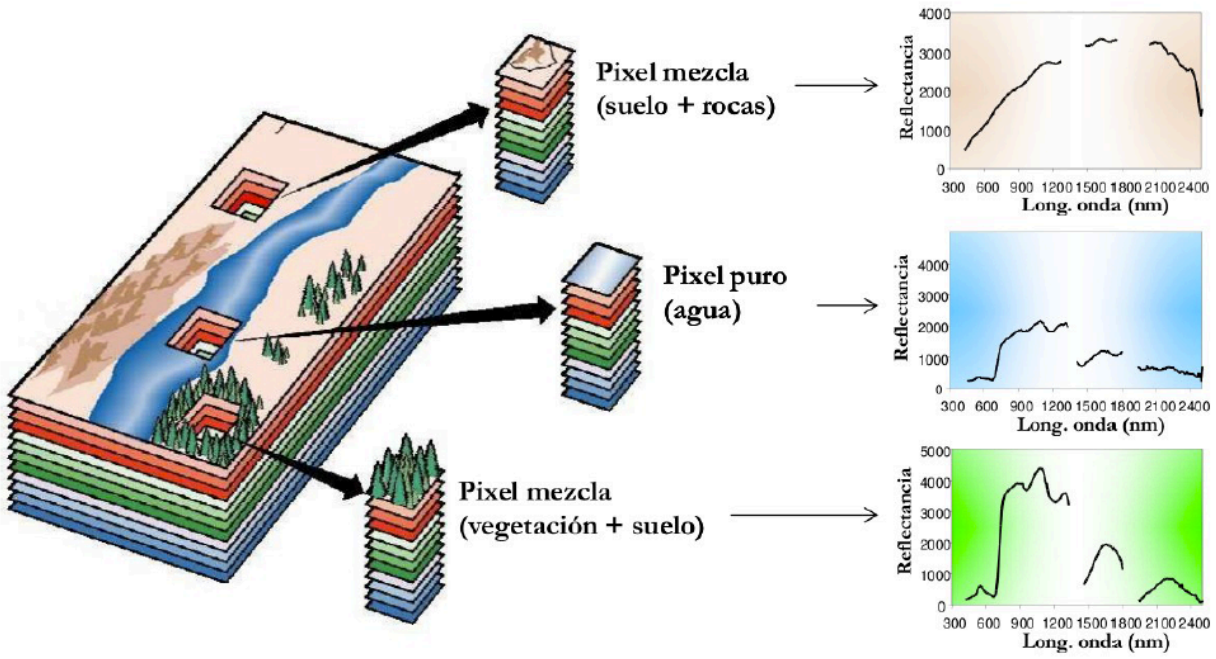


Figura 1.2. Tipos de píxeles en imágenes hiperespectrales.

Esta característica ha supuesto un descubrimiento muy importante en el procesamiento de las imágenes hiperespectrales y en concreto en los resultados obtenidos en su fase de clasificación, ya que mediante diferentes algoritmos que se mostrarán posteriormente se han podido determinar estos píxeles puros, que permiten realizar una clasificación más precisa que si se usaran firmas espectrales de librerías externas.

1.2. Técnicas de análisis hiperespectral y necesidad de paralelismo

La mayoría de las técnicas de análisis hiperespectral desarrolladas hasta la fecha presuponen que la medición obtenida por el sensor en un determinado píxel viene dada por la contribución de diferentes materiales que residen a nivel sub-píxel. El fenómeno de la mezcla puede venir ocasionado por una insuficiente resolución espacial del sensor, pero lo cierto es que este fenómeno ocurre de forma natural en el mundo real, incluso a niveles microscópicos, por lo que el diseño de técnicas capaces de modelar este fenómeno de manera adecuada resulta imprescindible. No obstante, las técnicas basadas

en este modelo son altamente costosas desde el punto de vista computacional. A continuación, detallamos las características genéricas de las técnicas basadas en este modelo y hacemos énfasis en la necesidad de técnicas paralelas para optimizar su rendimiento computacional.

1.2.1. Modelo lineal de la mezcla

En la actualidad, el modelo lineal de mezcla es el más utilizado en el análisis de imágenes hiperespectrales, debido a su sencillez y generalidad. Los píxeles mezcla [3] se representan como una combinación lineal de firmas asociadas a componentes espectralmente puros (llamados endmembers). Este modelo ofrece resultados satisfactorios cuando los componentes que residen a nivel sub-píxel aparecen espacialmente separados, situación en la que los fenómenos de absorción y reflexión de la radiación electromagnética incidente pueden ser caracterizados siguiendo un patrón estrictamente lineal.

El modelo lineal de mezcla puede interpretarse de forma gráfica en un espacio bidimensional utilizando un diagrama de dispersión entre dos bandas poco correlacionadas de la imagen, tal y como se muestra en la Figura 1.3. En la misma, puede apreciarse que todos los puntos de la imagen quedan englobados dentro del triángulo formado por los tres puntos más extremos (elementos espectralmente más puros). Los vectores asociados a dichos puntos constituyen un nuevo sistema de coordenadas con origen en el centroide de la nube de puntos, de forma que cualquier punto de la imagen puede expresarse como combinación lineal de los puntos más extremos, siendo estos puntos los mejores candidatos para ser seleccionados como endmembers [4]. El paso clave a la hora de aplicar el modelo lineal de mezcla consiste en identificar de forma correcta los elementos extremos de la nube de puntos N-dimensional. En la literatura reciente se han propuesto numerosas aproximaciones al problema de identificación de endmembers en imágenes hiperespectrales. En el siguiente capítulo se realizará una descripción de algunos de los algoritmos más destacados en esta función.

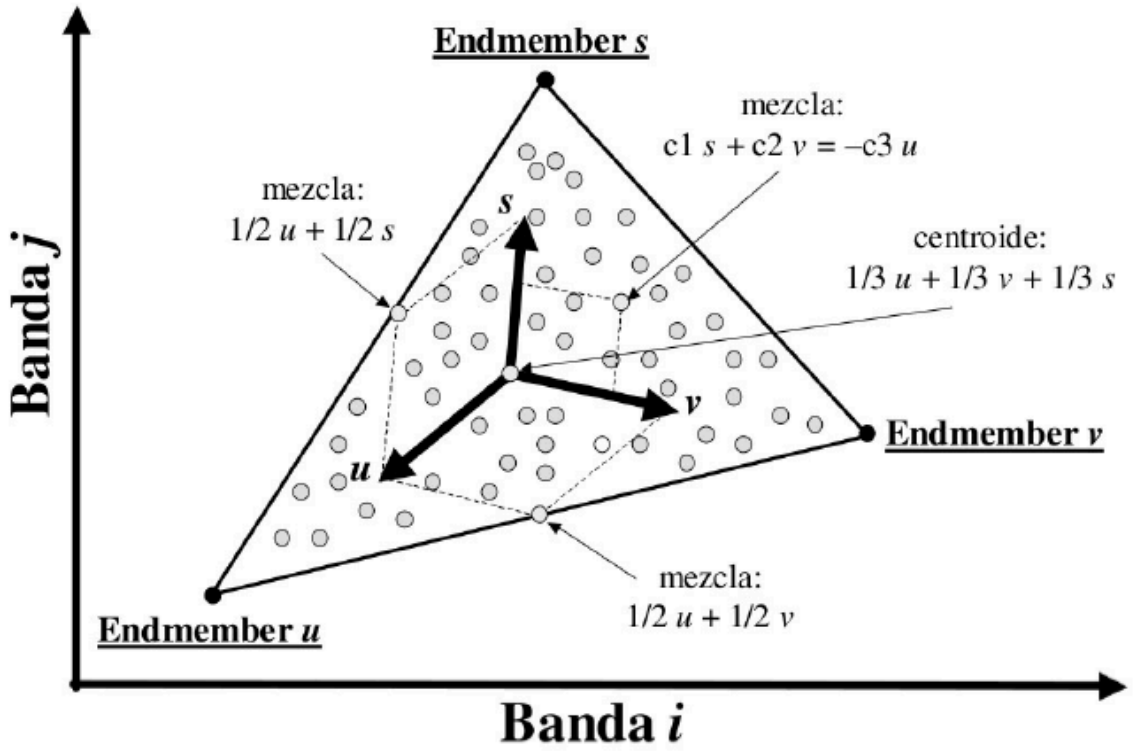


Figura 1.3. Interpretación gráfica del modelo lineal de mezcla.

1.2.2. Necesidad de paralelismo

Las técnicas de análisis hiperespectral anteriormente descritas se basan en la realización de operaciones matriciales que resultan muy costosas desde el punto de vista computacional [5]. Sin embargo, el carácter repetitivo de estas operaciones las hace altamente susceptibles de ser implementadas en diferentes tipos de arquitecturas paralelas, permitiendo así un incremento significativo de su rendimiento en términos computacionales y dotando a dichas técnicas de la capacidad de producir una respuesta rápida. Esta tarea es clave para la explotación de dichas técnicas en aplicaciones que requieren una respuesta en tiempo real o casi real.

Las técnicas de computación paralela han sido ampliamente utilizadas para llevar a cabo tareas de procesamiento de imágenes de gran dimensionalidad, facilitando la obtención de tiempos de respuesta muy reducidos y pudiendo utilizar diferentes tipos de arquitecturas [6-8]. En la actualidad, existen diferentes plataformas que permiten realizar una arquitectura paralela del diseño, tales como GPUs, que incorporan varios

procesadores, o hardware dedicado como las FPGAs y los circuitos integrados de aplicación específica (ASIC).

Las ventajas de implementación del algoritmo sobre hardware dedicado es la mayor eficiencia en la ejecución del algoritmo, y los reducidos tiempos de ejecución de los mismos además de una menor utilización de recursos. Sin embargo, es importante destacar que la complejidad de la tarea de implementación aumenta considerablemente, además de que el algoritmo original debe ser estudiado en detalle para encontrar diferentes ajustes o modificaciones, que no supongan una pérdida en cuanto a resultados, y una gran ganancia en cuanto a reducción de complejidad del algoritmo. En el capítulo 2 se mostrarán las modificaciones que se realizan sobre el algoritmo VCA, para poder ser implementados en hardware de forma eficiente.

1.3. Implementación hardware

Cuando se aborda el diseño de un sistema electrónico y surge la necesidad de implementar una parte con hardware dedicado, son varias las posibilidades que existen. En la figura 1.4 se muestra un esquema con las diferentes posibilidades que existen actualmente. En rojo se han marcado las FPGAs, porque es la que se empleará por los motivos que a continuación se explican.

Los circuitos full-custom también se denominan ASIC (Circuito Integrado para aplicaciones específicas), mientras que los circuitos estándar engloban a algunos procesadores y otros circuitos cuya lógica está ya fijada e implementada.

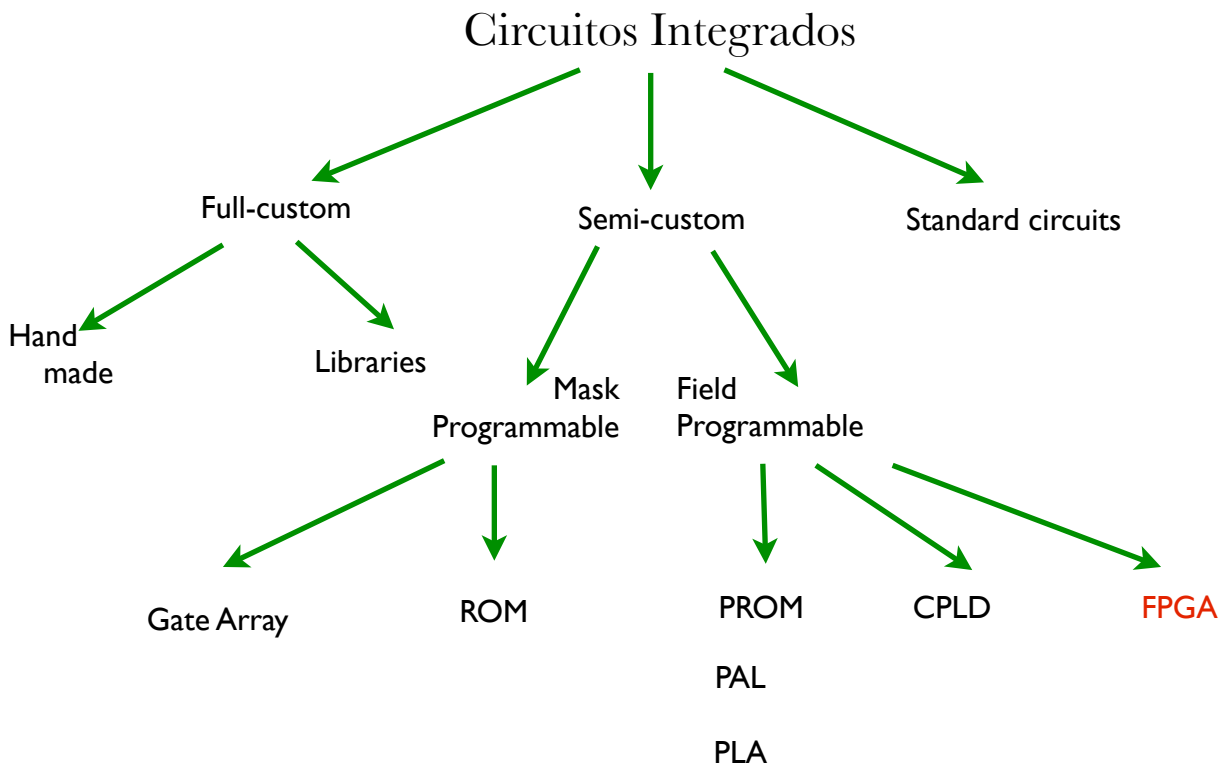
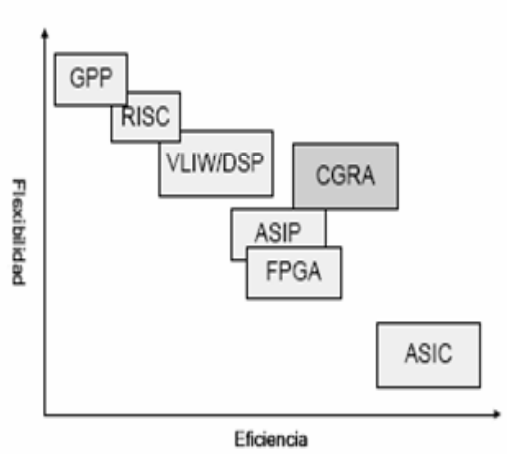


Figura 1.4. Esquema de los distintos tipos de circuitos integrados.

En la figura 1.5, se representa en un gráfico de eficiencia frente a flexibilidad algunos de los dispositivos pertenecientes a las tres familias principales anteriormente mencionadas.



GPP	*General Purpose Processor (Procesador de propósito general)*
RISC	***Reduced Instruction Set Computer***
VLIW	*Very Long Instruction Word*
DSP	***Digital Signal Processing***
ASIP	*Application Specific Instruction Processor*
FPGA	***Field Programmable Gate Arrays***
ASIC	*Application-Specific Integrated Circuits*
CGRA	***Coarse Grain Reconfigurable Array***

Figura 1.5. Flexibilidad vs eficiencia para diferentes arquitecturas.

Tal y como se observa, los procesadores ofrecen una flexibilidad muy elevada y la implementación de los programas sobre los mismos conlleva un bajo coste, resultando relativamente sencillo. No obstante la eficiencia es baja por lo que no es una opción válida en determinados casos en los que se desea alcanzar un nivel de prestaciones relativamente alto. Además, no se pueden utilizar técnicas de implementación en paralelo para la aceleración del proceso, más allá de las que ofrece el procesador en sí.

Los ASICs nos permiten realizar el diseño a partir de los elementos más comunes de la lógica digital. Es un circuito hecho a la medida de un uso particular y no es de propósito general. Se consigue una eficiencia muy alta, ya que el circuito se encuentra muy optimizado. No obstante, este tipo de implementación está dirigida a la etapa final de realización del producto, cuando se pretende comercializar y ya se han hecho todas las pruebas necesarias sobre un diseño muy robusto.

Las FPGAs son dispositivos programables, ideales para la realización de prototipos, en los que el número de cambios en la implementación es elevado, para poder ser sometido a diferentes pruebas una vez que ha sido programado en la placa. Además es un dispositivo que ofrece un gran equilibrio entre flexibilidad y eficiencia. Es por ello que se ha elegido en la implementación de los algoritmos que se presentan en capítulos posteriores.

En el siguiente apartado se explican las principales características de las FPGA.

1.3.1. FPGA (Field Programmable Gate Array)

Las FPGAs (Field Programmable Gate Arrays), introducidas por Xilinx en 1985, son dispositivos programables por el usuario, que como se mencionó anteriormente, presentan un buen equilibrio entre eficiencia y flexibilidad, lo que las convierte en una herramienta muy utilizada en diversas aplicaciones. Consisten en una matriz bidimensional de bloques configurables que se pueden conectar mediante recursos generales de interconexión, tal y como se representa en la figura 1.6. Estos recursos incluyen segmentos de pista de diferentes longitudes, más unos conmutadores

programables para enlazar bloques a pistas o pistas entre sí. En realidad, lo que se programa en una FPGA son los conmutadores que sirven para realizar las conexiones entre los diferentes bloques,más la configuración de los propios bloques.

El proceso de diseño de un circuito digital utilizando una matriz lógica programable puede descomponerse en dos etapas básicas:

1. Dividir el circuito en bloques básicos, asignándolos a los bloque configurables del dispositivo.
2. Conectar los bloques de lógica mediante los conmutadores necesarios.

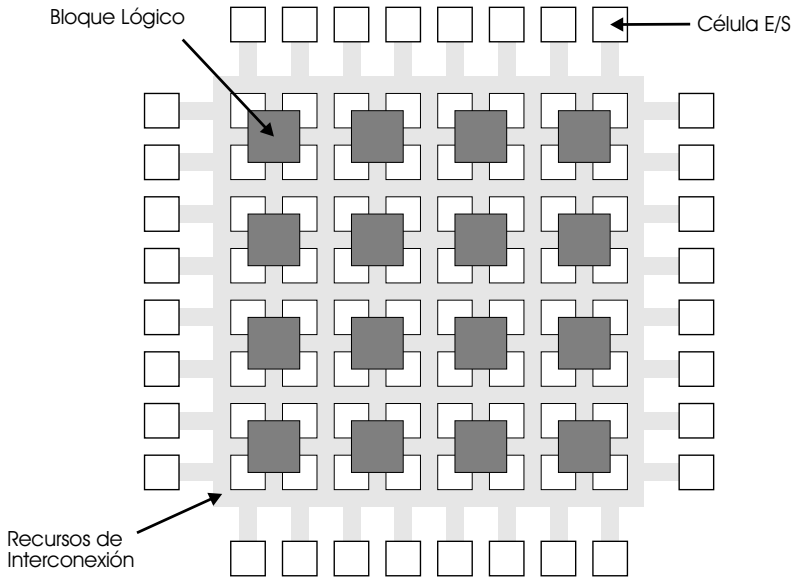


Figura 1.6. Estructura general de una FPGA.

Para ello el fabricante proporciona las herramientas de diseño adecuadas.

Los elementos básicos constituyentes de una FPGA, representados en la figura 1.8, son los siguientes:

1. Bloques lógicos, cuya estructura y contenido se denomina arquitectura. Hay muchos tipos de arquitecturas, que varían principalmente en complejidad (desde una simple puerta hasta módulos más complejos o estructuras tipo PLD). Suelen incluir biestables

para facilitar la implementación de circuitos secuenciales. Otros módulos de importancia son los bloques de Entrada/Salida.
2. Recursos de interconexión, cuya estructura y contenido se denomina arquitectura de enrutado.
3. Memoria RAM, que se carga durante el RESET para configurar bloques y conectarlos. (Existen dispositivos FPGA que se basan en antifusibles en vez de memorias RAM. Aunque la tecnología más extendida es la segunda).

Entre las numerosas ventajas que proporciona el uso de FPGAs dos destacan principalmente: el bajo coste de prototipado y el corto tiempo de producción. Sin embargo, no todo son ventajas. Entre los inconvenientes de su utilización están su baja velocidad de operación y baja densidad lógica (poca lógica implementable en un solo chip), frente a los dispositivos ASIC. Su baja velocidad se debe a los retardos introducidos por los conmutadores y las largas pistas de conexión.

Por supuesto, no todas las FPGA son iguales. Dependiendo del fabricante nos podemos encontrar con diferentes soluciones. Las FPGAs que existen en la actualidad en el mercado se pueden clasificar como pertenecientes a cuatro grandes familias, dependiendo de la estructura que adoptan los bloques lógicos que tengan definidos. Las cuatro estructuras se pueden ver en la figura 1.7, sin que aparezcan en la misma los bloques de entrada/salida.

1. Matriz simétrica, como son las de XILINX.
2. Basada en canales, ACMEL.
3. PLD jerárquica, ALTERA o CPLD’s de XILINX.
4. Mar de Puertas, ORCA.

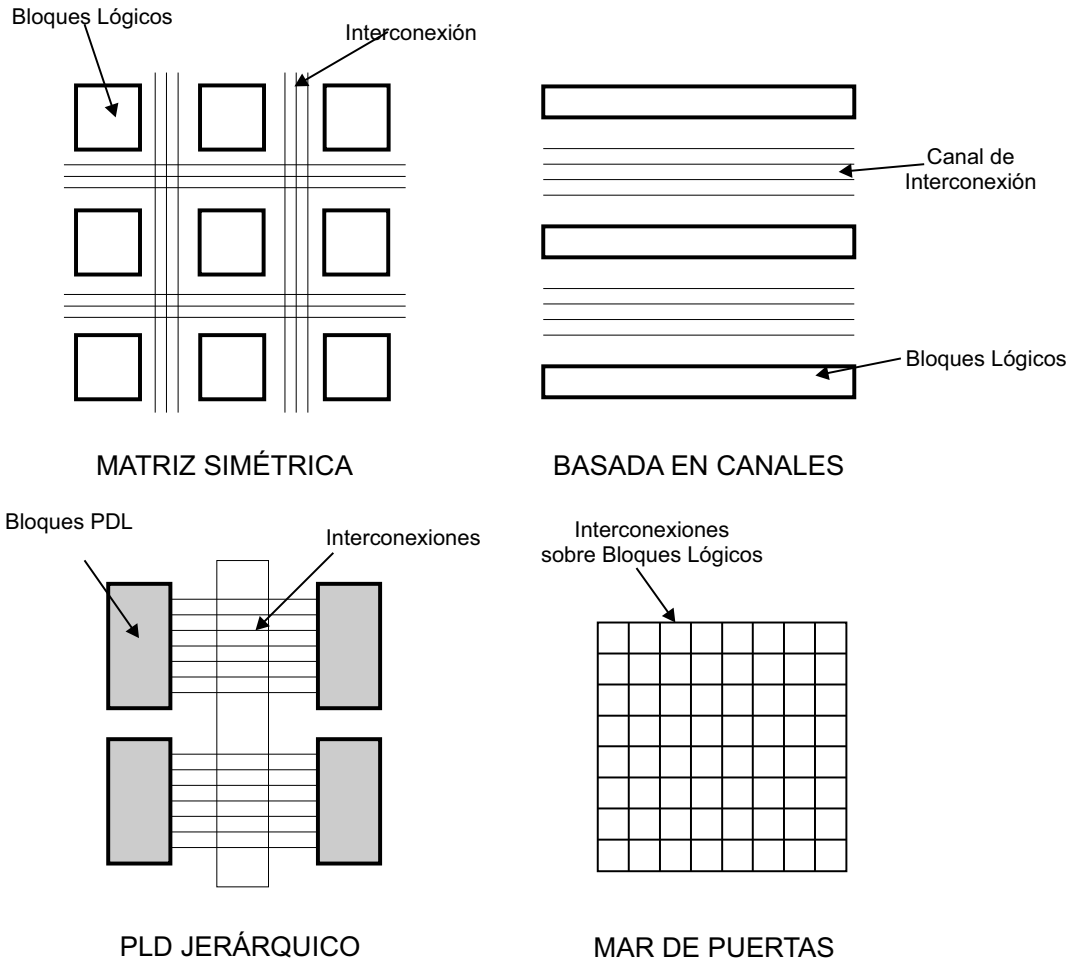


Figura 1.7. Tipos de FPGA.

En concreto, en este proyecto se sintetizarán los algoritmos sobre una FPGA Virtex 5 de Xilinx.

1.4. Objetivos y alcance del trabajo

Los objetivos principales de este Trabajo Fin de Máster se presentaron en la memoria del anteproyecto del mismo y se vuelven a desarrollar de manera más extendida a continuación.

1. Estudiar los algoritmos de extracción de endmembers más importantes, como son el Pixel Purity Index (PPI), el N-FINDR y el Vertex Component Analysis (VCA), y analizar las diferencias entre estos.

2. Realizar un estudio detallado del algoritmo VCA y plantear posibles modificaciones sobre el mismo, que por una parte simplifiquen la labor de implementación en lenguaje Verilog (lenguaje de bajo nivel) y por otra parte permitan obtener resultados de ejecución del algoritmo más eficientes. Para este último punto también será un requisito, analizar los posibles puntos a paralelizar del algoritmo.

3. Implementar el algoritmo VCA junto con las modificaciones introducidas sobre una FPGA Virtex-5, y obtener los resultados de síntesis y latencia, para poder establecer las comparativas pertinentes.

4. Exponer las principales conclusiones extraídas durante el desarrollo del trabajo, y mencionar posible mejoras y líneas de trabajo futuro, relacionados con el trabajo llevado a cabo y presentado en estas líneas.

1.5. Organización de la memoria

En este primer capítulo introductorio se han destacado una serie de ideas básicas que sirven al lector para entender los siguientes capítulos de la memoria que a continuación se presenta. En el capítulo que a continuación se aborda se presentan los diferentes algoritmos de procesamiento de imágenes hiperespectrales, haciendo énfasis en los algoritmos de extracción de endmembers y más concretamente en el algoritmo VCA. También en este capítulo se presentan las modificaciones que se llevan a cabo sobre el algoritmo.

En el tercer capítulo, se presentan las arquitecturas propuestas en este trabajo para la implementación del algoritmo VCA. Mediante un diagrama de bloques, se explican cada uno de los bloques implicados en cada una de las diferentes arquitecturas propuestas. Por otra parte, se muestra la metodología de verificación que se ha empleado, también haciendo uso de los pertinentes diagramas.

En el cuarto capítulo se presentan los resultados obtenidos, y por último, las principales conclusiones de los aspectos abordados en el trabajo y algunas posibles líneas futuras de investigación.

2.Algoritmos de procesamiento de imágenes hiperespectrales

Una vez vistos los conceptos fundamentales se tratan en este trabajo, se aborda un nuevo capítulo en el que se presentará el procesado que se lleva a cabo sobre las imágenes hiperespectrales, destacando los algoritmos de extracción de endmembers, los cuales juegan un papel fundamental.

En primer lugar se presentan las distintas etapas de procesado que se realizan sobre una imagen hiperespectral con el fin de obtener un resultado determinado. Se explica de manera resumida cada una de las etapas, así como los algoritmos más destacados dentro de las mismas.

Posteriormente el capítulo se centra en los algoritmos de extracción de endmembers principales, como son el PPI, el N-FINDR y el VCA, estableciendo comparaciones entre los mismos utilizando diversos parámetro como complejidad, prestaciones, etc.

Finalmente, se introducen las modificaciones que se han estudiado y puesto a prueba sobre el algoritmo VCA, para su implementación en hardware de manera más eficiente y menos compleja.

2.1. Etapas de procesamiento de imágenes hiperespectrales

El procesamiento de imágenes hiperespectrales se lleva a cabo en varias etapas. En la figura 2.1 se muestra un diagrama de bloques representando cada una de las etapas envueltas en el procesamiento.

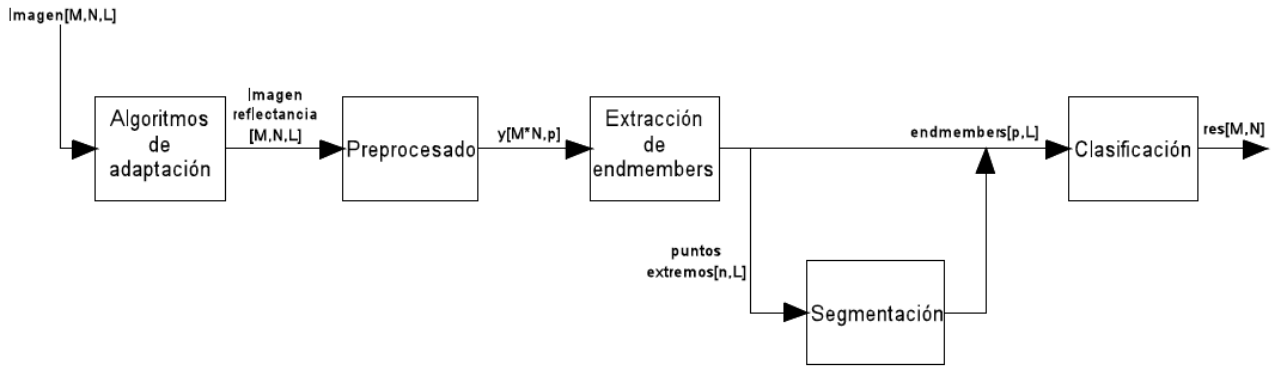


Figura 2.1. Diagrama de bloques de las etapas de procesamiento de imágenes hiperespectrales.

Como se puede observar, el proceso es relativamente complejo, involucrando varias etapas, aunque es importante resaltar que no todas ellas se deben aplicar en cada uno de los análisis hiperespectrales. Sin embargo, se ha comprobado que los resultados mejoran a medida que se aplican los procesos adecuados a la imagen.

En la primera etapa, se corrigen los datos numéricos aplicando ciertas técnicas, con el fin de poder trabajar con la imagen correctamente. La etapa de preprocesado plantea una disminución del número de bandas y por tanto de la dimensión de la imagen, a partir de una proyección de la misma sobre un espacio concreto que permite seleccionar aquellas componentes con mayor información.

La extracción de endmembers se ha convertido en un proceso fundamental, ya que la utilización de dichos endmembers en la etapa de clasificación proporciona resultados muy superiores, en comparación con la utilización de patrones extraídos de librerías externas. La segmentación se encuentra ligada a esta anterior ya que se implementa junto con algunos algoritmos de extracción de endmembers que producen más de un punto por material, y por tanto se les somete a una clusterización para agruparlos.

Cada una de las etapas se describe a continuación.

2.1.1. Algoritmos de adaptación

Esta etapa involucra todos los procesos necesarios desde que la imagen es capturada por el sensor hiperespectral, hasta que se lleva a cabo el procesamiento. En ella se incluyen los filtrados pertinentes a diferentes longitudes de onda, como la georreferenciación de los píxeles, es decir, la asignación de coordenadas UTM a cada uno de los puntos de la imagen, a partir de un fichero de coordenadas (esto sólo se realiza en las imágenes aéreas captadas por satélites o aeroplanos).

La adaptación más importante que se realiza sobre la imagen es la transformación de radiancia a reflectancia. La imagen capturada por el sensor es tomada en valores de radiancia, es decir que el sensor es excitado por la luz que incide sobre el objetivo a las diferentes longitudes de onda. Sin embargo, esta luz no siempre corresponde a la luz reflejada por la escena, sino que puede estar contaminada por radiación de la atmósfera, influida por la irradiación solar en el momento de captura de la imagen, además de verse afectada por la posición del sol con respecto a la vertical (zenith) y de la cámara. En la figura 2.2 se representan todos estos factores.

La corrección que se lleva a cabo, a partir de cierta información que se provee de la captura de la imagen, se realiza a partir de la siguiente relación:

$$\rho_\lambda = \frac{\pi L_\lambda d^2}{ESUN_\lambda \cos\theta_S}$$

donde:

ρ_λ: Reflectancia.

L_λ: Radiancia espectral en la apertura del sensor [$\frac{W}{m^2 sr \mu m}$].

d: Distancia del sol a la tierra (unidades astronómicas).

$ESUN_\lambda$: Media exoatmosférica (en el punto más alto de la atmósfera) de la irradiancia solar.

θ_S: Ángulo solar zenit (grados).

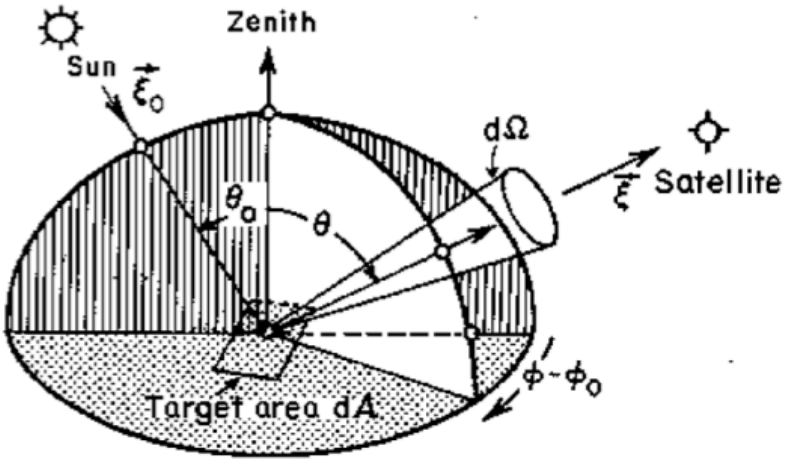


Figura 2.2. Agentes y parámetros que influyen en la adquisición de la imagen.

Los paquetes software dirigidos a trabajar con imágenes hiperespectrales (como es el caso de ENVI) realizan esta transformación a partir de especificar los datos que se mencionaron anteriormente, que como se comentó, varían según las condiciones de adquisición de la imagen.

2.1.2. Preprocesado

Los algoritmos de preprocesado están encaminados a reducir la dimensionalidad espectral a partir de realizar una transformación de espacio y seleccionar aquellas bandas que contienen mayor información.

Como se explicó en un principio, las imágenes hiperespectrales contienen una gran cantidad de datos. Sin embargo, se pueden buscar transformaciones de espacios matemáticos, para concentrar la información en pocas bandas y por tanto reducir el número de datos. Este es el caso de los algoritmos PCA (Principal Components Analysis) , SVD (Singular Value Decomposition) y MNF (Maximum Noise Fraction).

- Principal Components Analysis (PCA) [9-10]: La transformación de componentes principales genera combinaciones lineales de intensidades de los píxeles mutuamente incorreladas y que tienen máxima varianza. Es decir, se busca un nuevo espacio en el que cada una de las bandas está incorrelada con el resto y, por tanto, la matriz de covarianza sólo presenta valores no nulos en la diagonal. Si estos se ordenan de mayor a menor, se obtienen al principio, las bandas con mayor varianza y por lo tanto muy incorreladas con el resto.

- Singular Value Decomposition (SVD) [11]: es muy similar a la transformación anterior (PCA), pero en vez de buscar una proyección que tenga como restricción una minimización de mínimos cuadrados, se busca una proyección que maximize la potencia de la señal. En caso de que la media de todas las bandas sea nula, la transformada PCA y SVD producen el mismo resultado.

- Maximum Noise Fraction (MNF) [12]: El análisis de componentes principales, maximiza la varianza. No obstante, esto no siempre lleva a obtener la calidad deseada en la imagen (por ejemplo que el ruido sea mínimo). La transformación MNF, justamente trata de resolver ese problema, a partir de maximizar la relación señal-ruido (SNR), en vez de la varianza. Para ello se ejecutan dos transformaciones, una primera que trata de ecualizar el ruido en todas las bandas (realizando una transformación sobre la matriz de covarianza de ruido de la imagen original) y posteriormente sobre el resultado se implementa una transformación PCA.

2.1.3. Extracción de endmembers

Como se comentó en el primer capítulo de introducción del trabajo, el modelo matemático que más aceptación posee actualmente es el de existencia de píxeles puros y píxeles mezcla en la imagen (que son la mayoría), estando estos últimos formados por

una combinación de los anteriores. El modelo matemático de la composición lineal es el que sigue:

$$r = x + n = Ms + n$$

donde r es cada uno de los píxeles de la imagen, por tanto un vector de L componentes (siendo L el número de bandas espectrales). n es la componente de ruido aleatoria que existe en cada píxel, M es la matriz de endmembers:

$$M \equiv [m_1, m_2, ..., m_p]$$

y p el número de endmembers (o píxeles puros).

Para la utilización de este modelo, es imprescindible obtener los píxeles puros de la propia imagen, para lo cual se proponen numerosos algoritmos en la literatura, y múltiples variaciones de los mismos. Se destacan tres algoritmos fundamentalmente, cuyo objetivo es siempre el mismo, calcular los píxeles extremos de la nube de puntos que forma la imagen hiperespectral, empleando diversas metodologías:

- Pixel Purity Index (PPI) [13-14]: en este caso se proyecta la imagen un número de veces muy elevado (entre 1000 y 10000), sobre vectores aleatorios que se construyen, y el algoritmo, para cada proyección almacena al valor mayor de la proyección en valor absoluto. Aquellos píxeles que hayan salido más veces como mayores en las proyecciones se designan como puntos extremos, con estos se puede o bien realizar una clusterización y agruparlos en grupos que constituirán los endmembers, o realizar un proceso interactivo con una herramienta software que selecciona de manera gráfica dichos endmembers.

- N-FINDR [15]: busca un simplex de volumen máximo y un número de vértices p, basándose en la suposición de que en aquel simplex de mayor volumen que encierre a todos los puntos, sus p vértices serán los p endmembers buscados.

- Vertex Component Analysis (VCA) [16]: se obtiene cada uno de los endmembers al realizar una proyección completa de la imagen sobre un vector perpendicular al

anterior endmember obtenido, realizándose por tanto p proyecciones hasta encontrar p endmembers.

Todos los algoritmos mencionados serán explicados con más detalle posteriormente, al igual que serán expuestas sus principales diferencias.

2.1.4. Segmentación

Los algoritmos de segmentación establecen una serie de grupos o clusters, en función de unos datos de entrada. Pueden ser supervisados, en el caso en que se entrene al algoritmo a partir de cierta información que se posee de la imagen y por tanto el algoritmo la emplea en la separación, o no supervisados, en el caso en que no se entrene al algoritmo. Nunca se consigue eliminar por completo la supervisión, y es que al menos el sentido común humano siempre tiende a tomar una decisión en el resultado en base al conocimiento que se tiene del problema como es el caso que se muestra a continuación.

Uno de los algoritmos más empleados en la segmentación es el k-means [17]. En este caso se emplea de tal forma que como entrada del algoritmo se introducen los píxeles puros detectados anteriormente, ya que estos se quieren agrupar en clusters con características similares cuyos centros de gravedad se emplean para utilizarlos como patrón en la posterior clasificación. Por lo tanto, cuando se aplica el algoritmo k-means, se debe conocer el número de grupos k, en que se van a segmentar todos los puntos, ya que este es otro parámetro del algoritmo.

En la figura 2.3 se muestra un ejemplo de una segmentación realizada sobre los puntos que resultaron del algoritmo PPI en el tratamiento de una imagen, la agrupación de hace para 3 clusters. La representación en dos dimensiones es posible gracias a la función de Matlab mdscale, que pasa de un espacio n-dimensional a otro espacio m-dimensional. En este caso se ha pasado de 55 dimensiones a 2 dimensiones.

Los algoritmos de segmentación no se emplean siempre en el tratamiento de imágenes hiperespectrales, su empleo depende del algoritmo que se use para la obtención de endmembers. En general, estos algoritmos acompañan al algoritmo PPI, que

por su naturaleza puede generar más de un punto de un sólo endmember, y de ahí que se necesite una agrupación.

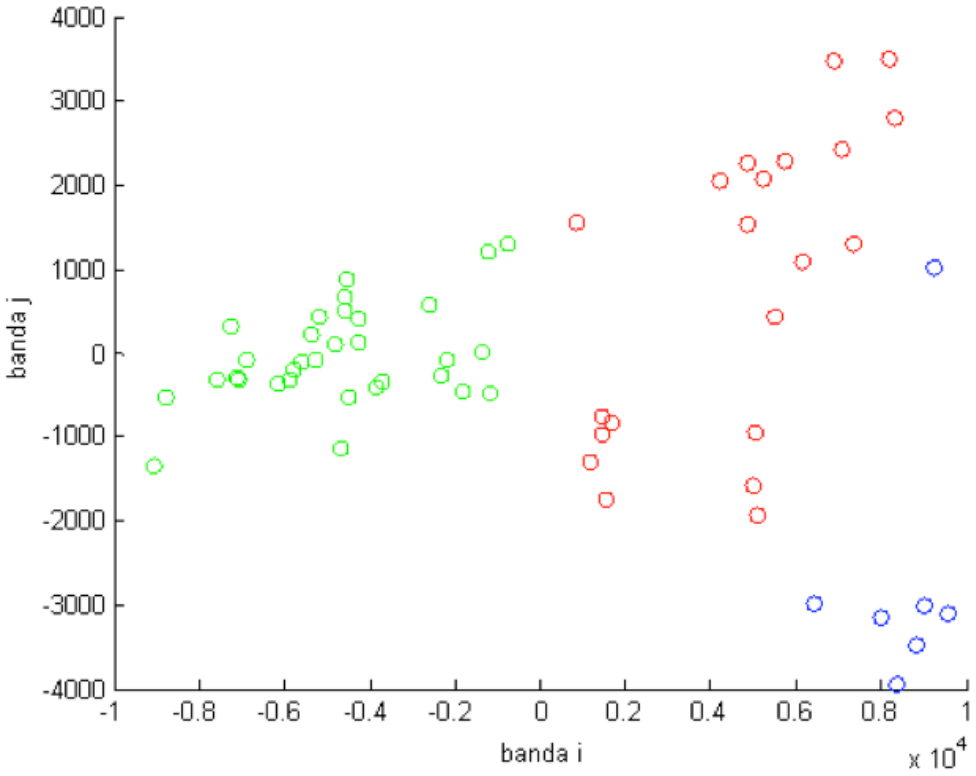


Figura 2.3. Segmentación k-means con 3 clusters.

2.1.5. Clasificación

En la clasificación (o *matching*) se parte de diferentes patrones, que bien pueden ser obtenidos de la propia imagen como se demostró en los pasos anteriormente citados (entonces estos patrones pasan a llamarse *endmembers*), o de librerías externas, de las diferentes webs de centros de investigación [18-19] que toman el espectro de varios materiales (agua, asfalto, sal, etc.) a partir de análisis con espectrómetros.

Las métricas de comparación entre píxeles son múltiples (en la bibliografía se proponen varias [20-21]), de las que se destacan tres principales por su sencillez a la hora de implementar y sus buenos resultados.

- Spectral angle (Ángulo espectral): se basa en calcular el ángulo existente entre cada pixel de la imagen y el patrón empleado.

- Cross-correlation (coeficiente de Pearson): calcula el coeficiente de Pearson (correlación) entra cada uno de los píxeles de la imagen y el patrón.

- Matched filter: se trata de, empleando un patrón concreto, realizar una proyección, eliminando el ruido no deseado, de los píxeles de la imagen sobre el patrón, para conocer la presencia del mismo sobre cada uno de ellos.

Todos ellos presentan como resultado una matriz en escala de grises para cada uno de los patrones empleados.

2.2. Algoritmos de extracción de endmembers

En este subapartado se analizan en detalle los principales algoritmos de extracción de endmembers, PPI, N-FINDR y VCA.

2.2.1. Pixel Purity Index

El algoritmo PPI pertenece al conjunto de los métodos interactivos y es el más representativo. Su objetivo es localizar los puntos espectralmente más puros de la imagen hiperespectral, basándose en la suposición de que los puntos más extremos del conjunto de puntos son los mejores candidatos para ser utilizados como endmembers. Los parámetros de entrada del algoritmo son el número de iteraciones a realizar y el valor umbral para seleccionar píxeles puros. El funcionamiento del algoritmo se describe en los siguientes pasos:

1. En primer lugar, el algoritmo asigna un índice de pureza a todos los píxeles de la imagen. El contador de cada punto se inicializa al valor 0.
2. Seguidamente, se genera un vector unitario aleatorio, que recibe el nombre de skewer o “divisor”. El objetivo de este vector es particionar el conjunto de puntos, como veremos a continuación.

3. El tercer paso consiste en proyectar todos los puntos de la imagen hiperespectral sobre el vector unitario antes generado, identificando los puntos extremos en la dirección definida por el vector unitario. El índice de pureza de los puntos extremos se incrementa en una unidad.
4. Los pasos 2-3 del algoritmo se repiten tantas veces como el usuario especifique en el parámetro de entrada, número de iteraciones.
5. Tras la ejecución de un número amplio de iteraciones, se obtiene como resultado una imagen de pureza formada por los índices asociados a cada uno de los píxeles de la imagen.
6. Utilizando el valor umbral especificado como parámetro, se seleccionan los puntos de la imagen cuyo índice de pureza asociado supera dicho valor umbral. Estos puntos son etiquetados como “puros”.

El método PPI contiene etapas totalmente automatizadas, como la fase de generación de la imagen de pureza, pero es necesaria una etapa final, que puede ser o bien altamente interactiva, en la que el usuario selecciona manualmente los píxeles que quiere utilizar como endmembers, o una clusterización como se explicó anteriormente. En ambos casos el usuario no conoce a priori cuál es el número apropiado de endmembers, por lo que debe escoger el número de endmembers en base a su intuición. Este hecho pone de manifiesto la conveniencia de cierto conocimiento a priori sobre la imagen. Esta característica, unida a otras como la aleatoriedad en el proceso de generación de vectores unitarios, representan los principales inconvenientes de esta metodología [22]. Por otra parte, es importante destacar que el método PPI puede generar endmembers artificiales en caso de que el usuario del método seleccione conjuntos de puntos en el proceso interactivo de identificación de firmas espectrales extremas.

En la figura 2.4 se muestra una imagen representativa del método PPI, en el que como se observa, se produce la proyección de los puntos de la imagen sobre los vectores aleatorios, seleccionándose aquellos que están más distanciados, es decir, que son más extremos en cada caso.

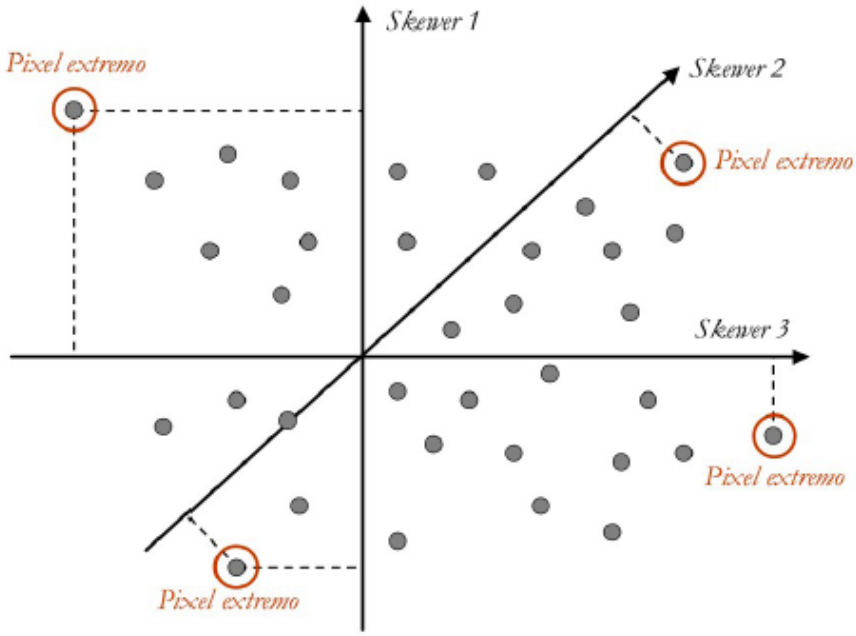


Figura 2.4. Representación gráfica del algoritmo PPI.

2.2.2. N-FINDR

El algoritmo N-FINDR realiza el cálculo de los endmembers de manera simultánea. Son múltiples las modificaciones que se han presentado sobre el algoritmo, y que mejoran considerablemente su ejecución, sin embargo en lo que sigue se presentarán las ideas fundamentales del mismo.

Antes de realizar la ejecución del algoritmo es fundamental llevar a cabo la fase de preprocesado que incluye la determinación del número de endmembers a calcular, p, y la transformación espacial de la imagen, para realizar una reducción dimensional espectral de la misma de L a p bandas. Una vez se ha ejecutado este preprocesado, se presenta el paso principal del algoritmo:

1. Se ejecuta una búsqueda exhaustiva que consiste en, para un conjunto arbitrario p de vectores e_1, e_2, ..., e_p calcular el volumen generado por el simplex (e_1, e_2, ..., e_p), a partir de la siguiente expresión:

$$V(e_1,...,e_p) = \frac{\det\begin{bmatrix} 1 & 1 & ...1 \\ e_1 & e_2 & ...e_p \end{bmatrix}}{(p-1)!}$$

Se destaca que la búsqueda necesita de $\binom{N}{p} = (N!)/(p!(N-p)!)$ iteraciones, por lo que el coste del algoritmo es muy elevado. Una vez encontrado el conjunto de vectores que maximizen el volumen, se concluye el proceso, y ese conjunto de vectores es el conjunto de p endmembers buscados. Obsérvese como se comentó en un principio que el algoritmo determina los endmembers de manera simultánea.

Tal y como se mencionó en el punto 2 de la ejecución del algoritmo, se trata de un proceso muy complejo y costoso, prácticamente inviable de ser llevado a la práctica y por tanto muchos esfuerzos han sido puestos en reducir estos aspectos [23]. Una de las principales líneas de investigación se basan en buscar una parada mucho más temprana del algoritmo, una vez que una iteración y la anterior produzcan resultados muy parecidos. Para ello se inicializa el N-FINDR con vectores aleatorios en numerosas ocasiones, y por el eso el algoritmo recibe el nombre de Random N-FINDR (RN-FINDR).

2.2.3. Vertex Component Analysis

El algoritmo VCA trata de descomponer espectralmente y de manera lineal los vectores mezcla de la imagen en sus correspondientes componentes puras (endmemebers). Se trata de un algoritmo no supervisado, que se basa en dos aspectos, por un lado los endmembers son los vértices de un simplex, y la transformación afin de un simplex es también otro simplex. El algoritmo funciona tanto con imágenes proyectadas a otro espacio y reduciendo la dimensión espectral de la imagen, como con las imágenes originales que no sufren un preprocesado previo.

El método consiste en que de manera iterativa se proyectan los valores de la imagen sobre una dirección perpendicular al subespacio determinado por los

endmembers ya calculados. El nuevo endmember se determina como el valor extremo de esta proyección. Se producen tantas iteraciones como endmembers se deseen calcular, es decir, p iteraciones. Este hecho supone que computacionalmente es mucho menos costoso que los dos algoritmos anteriores tal y como se verá posteriormente.

En la figura 2.5 se representa graficamente el funcionamiento del algoritmo VCA, y en la figura 2.6 un ejemplo del algoritmo para una nube de puntos, y tres endmembers.

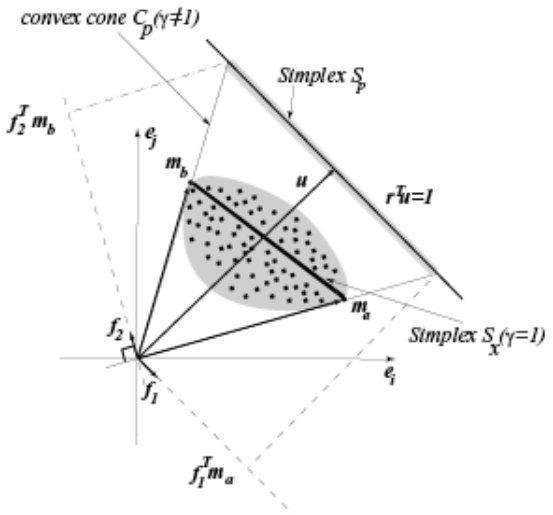


Figura 2.5. Representación gráfica del algoritmo VCA.

La nube de puntos que conforman la imagen, en caso de ausencia de ruido en la misma, se encuentran contenidos en un cono convexo C_p, a su vez perteneciente al subespacio E_p de dimensión p. El algoritmo VCA se encarga de determinar este subespacio E_p mediante SVD y proyectar los puntos del cono (la imagen) sobre este subespacio, $y=r/(r^T u)$, formándose un simplex S_p. Este simplex se encuentar contenido en un espacio de dimensión $p-1$. En el caso de que la proyección se realizara sobre un subespacio E_d de mayor dimensión $p \leq d \leq L$, la proyección de C_p sobre E_d, seguido de una proyección adecuada es también un simplex con los mismos vértices.

Si la relación señal ruido disminuye, el factor de escala que se aplicaba en el caso anterior, $r/(r^T u)$, aumentaría el ruido presente en la imagen, que en este caso es superior. Por este motivo, es preferible no usar este factor de escala, y determinar el

nuevo espacio sobre el que se quiere proyectar la imagen, mediante el método PCA. En este caso, el nuevo espacio será de dimensiones `p-1`.

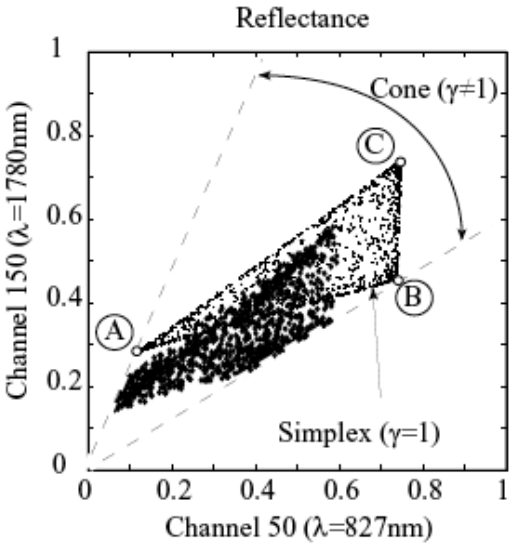


Figura 2.6. Scatterplot en dos dimensiones de píxeles mezcla formados por 3 endmembers.

El algoritmo completo se muestra en las siguientes líneas:

Algorithm 1 : Vertex Component Analysis (VCA)

INPUT: p, $\mathbf{R} \equiv [\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N]$

$SNR_{th} = 15 + 10\log_{10}(p)$ dB
if $SNR > SNR_{th}$ **then**
  $d := p$;
  $\mathbf{X} := \mathbf{U}_\mathbf{d}^\mathbf{T}\mathbf{R}$; {$\mathbf{U_d}$ obtained by SVD}
  $\mathbf{u} :=$ mean $(\mathbf{X})$; {$\mathbf{u}$ is a $1 \times d$ vector}
  $[\mathbf{Y}]_{:,j} := [\mathbf{X}]_{:,j}/([\mathbf{X}]_{:,j}^T\mathbf{u})$; {projective projection}
else
  $d := p - 1$;
  $[\mathbf{X}]_{:,j} := \mathbf{U}_\mathbf{d}^\mathbf{T}([\mathbf{R}]_{:,j} - \overline{\mathbf{r}})$; {$\mathbf{U_d}$ obtained by PCA}
  $c := \arg\max_{j=1\ldots N} \|[\mathbf{X}]_{:,j}\|$;
  $\mathbf{c} := [c\,|\,c\,|\ldots|\,c]$; {$\mathbf{c}$ is a $1 \times N$ vector}
  $\mathbf{Y} := \begin{bmatrix}\mathbf{X} \\ \mathbf{c}\end{bmatrix}$;
end if

14: $\mathbf{A} := [\mathbf{e}_u \,|\, \mathbf{0} \,|\, \ldots \,|\, \mathbf{0}]$; {$\mathbf{e}_u := [0, \ldots, 0, 1]^T$ and $\mathbf{A}$ is a $p \times p$ auxiliary matrix}
15: **for** $i := 1$ to p **do**
16: $\mathbf{w} :=$ randn $(0, \mathbf{I}_p)$; {$\mathbf{w}$ is a zero-mean random Gaussian vector of covariance $\mathbf{I}_p$}
17: $\mathbf{f} := \frac{(\mathbf{I} - \mathbf{A}\mathbf{A}^{\#})\mathbf{w}}{\|(\mathbf{I} - \mathbf{A}\mathbf{A}^{\#})\mathbf{w}\|}$; {$\mathbf{f}$ is a vector orthonormal to the subspace spanned by $[\mathbf{A}]_{:,1:i-1}$.}
18: $\mathbf{v} := \mathbf{f}^T\mathbf{Y}$;
19: $k := \arg\max_{j=1,\ldots,N} |[\mathbf{v}]_{:,j}|$; {find the projection extreme.}
20: $[\mathbf{A}]_{:,i} := [\mathbf{Y}]_{:,k}$;
21: $[indice]_i := k$; {stores the pixel index.}
22: **end for**
23: **if** $SNR > SNR_{th}$ **then**
24: $\widehat{\mathbf{M}} := \mathbf{U}_\mathbf{d}[\mathbf{X}]_{:,indice}$; {$\widehat{\mathbf{M}}$ is a $L \times p$ estimated mixing matrix}
25: **else**
26: $\widehat{\mathbf{M}} := \mathbf{U}_\mathbf{d}[\mathbf{X}]_{:,indice} + \overline{\mathbf{r}}$; {$\widehat{\mathbf{M}}$ is a $L \times p$ estimated mixing matrix}
27: **end if**

Las líneas de la 1 a la 13 realizan una proyección de la matriz de entrada (imagen) a un espacio calculado o bien empleando la transformación SVD a un espacio de `p` dimensiones o bien la transformación PCA a un espacio de `p-1` dimensiones, en función de que la relación señal ruido (SNR) de la misma sea o no superior a un umbral, que se calcula en la primera instrucción del algoritmo. Esta primera fase que se ha descrito constituye la fase de preprocesado de la imagen. Destacar que los pasos 4 y 9 aseguran que ninguno de los productos escalares entre `[X]:,i` y `u` sean negativos, aspecto crucial para el correcto funcionamiento del algoritmo VCA.

A partir de la línea 14, comienza el algoritmo VCA con la inicialización de la matriz `A` , matriz sobre la que se irán almacenando las proyecciones de los endmembers que se vayan encontrando, de la forma en que se muestra, todos sus columnas a cero, exceptuando la primera que está ocupada por el vector $e_u = [0,0,...,1]'$. Esta inicialización de la matriz `A` produce la reducción de una de las dimensiones, obteniéndose todos los puntos proyectados en un espacio de `p-1` dimensiones. Este hecho es fundamental, ya que se recuerda que el simplex se obtiene proyectando los datos sobre un espacio de estas características.

Una vez inicializada la matriz, se entra en el bucle, línea 15, que calcula en cada iteración un vector `f`, ortonormal a la matriz `A`, a través de la creación de un vector aleatorio, que es de esta naturaleza para evitar realizar una proyección sobre las columnas de la matriz `A` de un vector nulo. Una vez calculado `f`, en la línea 18 se realiza la proyección de la matriz `Y` sobre este vector, y se almacena en `v`, de tal forma que en la siguiente línea se calcula el índice del valor mayor absoluto del vector, y se almacena en `k`.

En la línea 20, con este índice calculado, se selecciona de la matriz `Y` el vector, que supondrá la proyección del nuevo endmember, y posteriormente se guarda el índice `k` en un vector de p componentes.

Las líneas 23 a 27 se encargan de reconstruir la imagen al espacio original de `L` bandas.

2.2.4. Comparativa entre algoritmos

En este apartado se pretende realizar una comparativa de los tres algoritmos presentados anteriormente, basándonos en dos aspectos, la bondad del método en cuanto a precisión de los resultados obtenidos, y el coste computacional de los algoritmos. Estas comparaciones han sido extraídas de [VCA algorithm].

Para realizar la comparativa en cuanto a la precisión de los resultados obtenidos se emplean tres ángulos que se calculan a partir del método spectral angle. Se destaca que estos ángulos se pueden determinar ya que se trata de imágenes artificiales, donde se conoce la firma espectral pura que se debe obtener y por tanto se puede comparar con la que proporciona el algoritmo. Las expresiones de los ángulos son las siguientes:

$$\theta_i \equiv (arc\cos\frac{\left\langle m_i, \hat{m}_i \right\rangle}{\left\| m_i \right\| \left\| \hat{m}_i \right\|})$$

que determina el ángulo espectral que existe entre el endmember estimado $\hat{m}_i$ y el endmember real m_i .

$$\beta_i \equiv (arc\cos\frac{\left\langle [S]_{i,:},[\hat{S}]_{i,:}\right\rangle}{\left\|[S]_{i,:}\right\|\left\|[\hat{S}]_{i,:}\right\|})$$

determinándose las diferencias entre las abundancias de cada píxel estimadas y las reales, siendo la fórmula para el cálculo de las abundancias $\hat{S} = \hat{M}^{\#}[r_1, r_2, ..., r_N]$, que representa la pseudoinversa de la matriz de endmembers multiplicado por cada uno de los píxeles de la imagen.

Finalmente la última medida que se emplea es *spectral information divergence* (SID), para comparar la similitud de dos firmas.

$$SID_{m_i,\hat{m}_i} \equiv D(m_i \mid \hat{m}_i) + D(\hat{m}_i \mid m_i)$$

$$D(m_i \mid \hat{m}_i) \equiv \sum_{j=1}^{L} p_j \log\left(\frac{p_j}{q_j}\right)$$

siendo $p_j = m_{ij} / \sum_{k=1}^{L} m_{ik}$ y $q_j = \hat{m}_{ij} / \sum_{k=1}^{L} \hat{m}_{ik}$.

Todas estas medidas producen p valores (tantos como endmembers se calculen) y por tanto para la representación gráfica se calcula el rms del vector de valores. En la figura 2.7 se muestran los resultados producidos por los tres algoritmos, en función de la relación señal ruido que existe en el conjunto de datos.

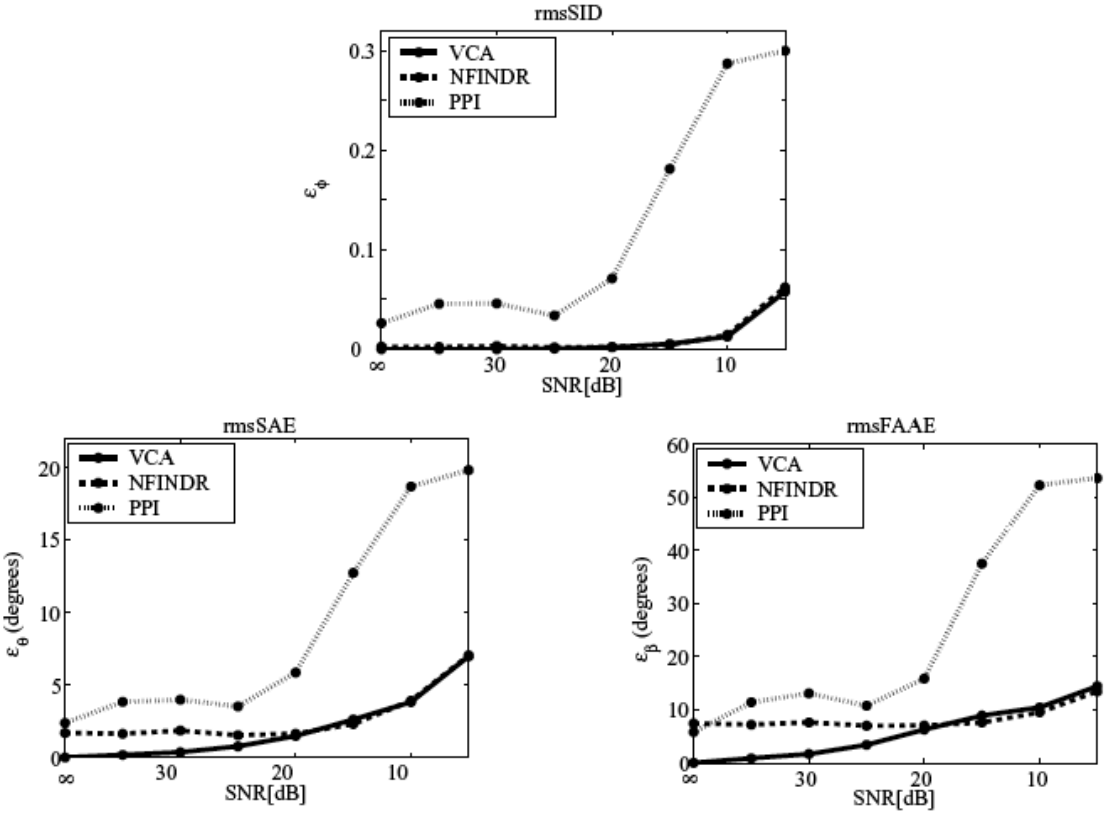


Figura 2.7. Arriba: rmsSID. Abajo izquierda: rmsSAE. Abajo derecha: rmsFAAE. Parámetros del conjunto de datos (N=1000, p=3, $\mu_1=\mu_2=\mu_3=1/3, ß_1=20, ß_2=1$).

Las gráficas muestran como en general, los resultados producidos por el VCA y N-FINDR se aproximan a medida que el ruido en la imagen aumenta, o lo que es lo mismo, la relación señal ruido disminuye, sin embargo para valores altos de SNR, el VCA es el que mejores resultados proporciona. Por otra parte, destacar que los resultados proporcionados por el PPI distan bastante de los que producen los otros dos algoritmos.

A continuación se analiza gráficamente la complejidad de los algoritmos, que se representa en la figura 2.8. En un caso se varía el número de endmembers a detectar, y en el otro el número de píxeles de la imagen.

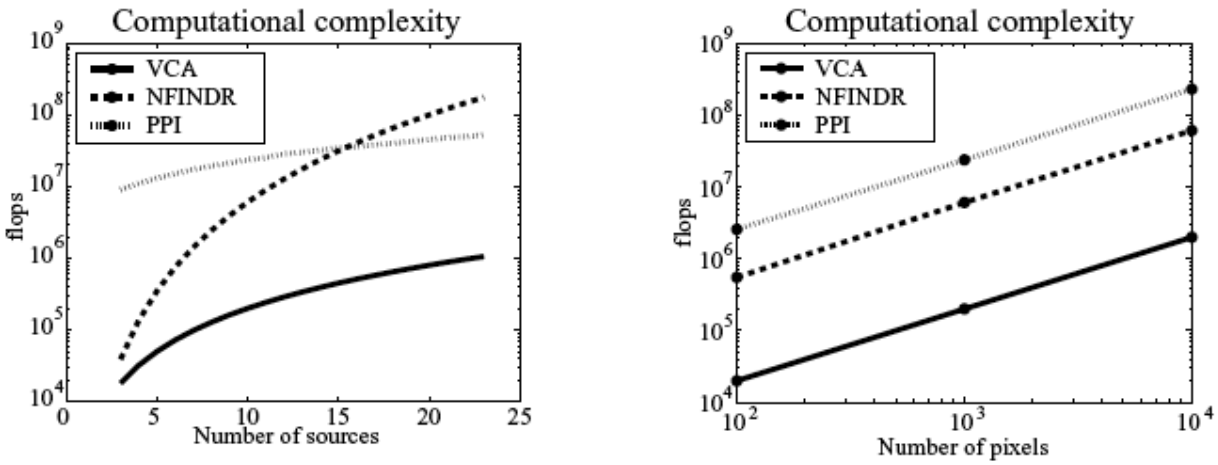


Figura 2.8. Complejidad computacional medido en número de operaciones en punto flotante. Izquierda: con respecto al número de endmembers. Derecha: con respecto al número de píxeles.

Como se observa, la complejidad del VCA es menor en todos los casos, siendo incluso para 5 endmembers un orden de magnitud inferior al N-FINDR.

2.3. Modificaciones sobre el algoritmo VCA

A la hora de implementar el algoritmo en hardware, en primer lugar hay que plantearse la cuestión de cuáles son los posibles modificaciones que se pueden realizar en el algoritmo, para que el coste computacional disminuya y a su vez no se produzca una pérdida considerable de información en los datos que se manejan o el proceso que afecten notoriamente al resultado.

Existen algunas modificaciones generales que se llevan a cabo en el algoritmo VCA y que reducen considerablemente el coste computacional. La parte del algoritmo en que se realiza el preprocesado de la imagen no se modifica, ya que no va a ser implementada en hardware. Por tanto, como entrada al algoritmo, se tomará la imagen preprocesada `Y`. A partir de la línea 14 comienza el código VCA, que fue descrito anteriormente. Se repasan algunos detalles relativos a la complejidad de la implementación en hardware.

La línea 16 establece que el vector `w`, que se emplea para obtener el vector perpendicular al subespacio generado por `A`, es un vector Gausiano aleatorio de media

cero. Dada la lógica del problema, este vector puede ser un vector fijo durante todo el algoritmo, siempre y cuando sus componentes sean distintas de cero.

La línea 17 presenta un punto crítico a resolver en hardware, se trata del cálculo de la pseudoinversa, que no sólo atañe complicaciones en cuanto al diseño del algoritmo en hardware, sino que significa un coste computacional elevado. Además en esta línea se realiza una normalización del vector para que este sea un vector ortonormal. Este hecho no es necesario, ya que el algoritmo devuelve como resultado el índice de aquella proyección cuyo valor absoluto sea mayor, si realizamos una normalización, lo que se está haciendo es dividir a todas las componentes de `f` por un mismo valor y por tanto afectar por igual a todos los productos que se hacen en la proyección, de tal forma que el índice del ganador no cambia.

Una vez presentado el análisis del algoritmo VCA original, se introduce el algoritmo que se ha desarrollado con diferentes modificaciones que tratan de solventar los problemas planteados y posteriormente se explican los cambios que se llevan a cabo.

Algorithm: Vertex Component Analysis (VCA) Modifications

INPUT: `p, Y=[y1,y2,...,yN]` {`Y` is the projected image calculated in the preprocess}

1. `A := [eu|0| ... |0]` ;{e_u := `[0, ..., 0,1]`^{T} and `A` is a `px(p+1)` matrix}
2. U_b := `[0| ... |0]` ;{U_b is a `pxp` matrix}
3. `w := [1,...,1]` ;{`w` is a `px1` vector}
4. f_{acc} := `[0,...,0]`;
5. `[r,s] := arg min` $_{j=1,...,p;\ i=1,...,N}$ $|[Y]_{j,i}|$ {minimum element `Y`}
6. `bin1 := fp2bin(`$[Y]_{r,s}$`);`
7. `Y := Y/2^(bin2dec(`$[bin1]_{2:9}$`)-127);`
8. **for** `i:= 1` **to** `p` **do**
9. $[U_b]_{:,i}$:= $[A]_{:,i}$;
10. **for** `j:= 3` **to** `i` **do**
11. $c_1(i,j-1) := \frac{[A]_{:,i}^T [U_b]_{:,j-1}}{[U_b]_{:,j-1}^T [U_b]_{:,j-1}}$;
12. $[U_b]_{:,i} := [U_b]_{:,i} - c_1 * [U_b]_{:,j-1}$;

13. **end for**

14. $c_2(i) := \frac{w^T [U_b]_{:,i}}{[U_b]_{:,i}^T [U_b]_{:,i}}$;

15. $f_{acc} := f_{acc} + c2 * [U_b]_{:,i}$;
16. f := w – f_{acc}; {f is a ortogonal vector to the subspace spanned by $[A]_{:,1:i}$}
17. **if** i == 1 **then**
18. f_{acc} := [0,...,0]; {Reinitialize the accumulator}
19. **else**
20. m := arg $\min_{j = 1, \ldots, p} |[f]_j|$ {minimum element of f}
21. bin2 := fp2bin($[f]_m$);
22. f := f/2^(bin2dec($[bin2]_{2:9}$)-127);
23. **end if**
24. v := round(f^T)*round(Y);
25. k := arg $\max_{j = 1, \ldots, N} |[v]_{:,j}|$ {find the projection extreme}
26. $[A]_{:,i+1}$:= $[Y]_{:,k}$;
27. $[index]_i$ = k;
28. **end for**

Comencemos por los parámetros de entrada al algoritmo, el parámetro p es el número de endmembers a calcular, y la matriz Y, es la imagen proyectada sobre el espacio calculado E_p, en el preprocesado. Cada uno de los vectores que se presentan en la imagen Y, y_1, y_2, ..., y_N, es de dimensión p, hay N vectores, que es el número de píxeles de la imagen. En este punto es importante destacar que la matriz Y sufre el mismo proceso de normalización que el vector f, escrito entre las líneas 20-22, y que se puede apreciar en las líneas 5-7.

En las primeras 4 líneas se inicializan matrices y vectores. La matriz A en este caso, a diferencia del VCA original presenta p+1 columnas, ya que el vector de inicialización, e_u se mantiene en la matriz y por tanto se necesita una columna más. La matriz U_b se emplea para almacenar los vectores de la base ortogonal que se crea para posteriormente calcular el vector f. Obsérvese que el vector w, tal y como se comentó

anteriormente, queda fijado durante todo el algoritmo con todas sus componentes iguales a la unidad.

De las líneas 9 a la 13, se calcula la nueva base mencionada, y que es fundamental para evitar tener que calcular la pseudoinversa. Por lo tanto se modifica el cálculo de `f` que ahora se realiza utilizando el método de ortogonalización de Gram-Schmidt, que trata de generar una base ortogonal y utilizando esta misma, generar un vector ortogonal a la propia base. Véase las siguientes ecuaciones que describen el método.

Supongamos que se han determinado los endmembers a_1, a_2 y a_3, el cálculo de la base se realiza como sigue:

$$u_1 = a_1$$

$$u_2 = a_2 - \frac{\langle a_2, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1 = a_2 - c_1(2,1)u_1$$

$$u_3 = a_3 - \frac{\langle a_3, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1 - \frac{\langle a_3, u_2 \rangle}{\langle u_2, u_2 \rangle} u_2 = a_3 - c_1(3,1)u_1 - c_1(3,2)u_2$$

Una vez se dispone de la base ortogonal, se calcula en cada iteración el vector perpendicular a dicha base, sobre el que se proyecta la imagen:

$$f = w - \sum_{j=1}^{j=i} \frac{\langle w, u_j \rangle}{\langle u_j, u_j \rangle} u_j = w - \sum_{j=1}^{j=i} c_2(j)u_j$$

En la línea 9 del algoritmo, se asigna el nuevo endmember calculado al vector columna de la base U_b de la iteración correspondiente, y posteriormente se entra en el bucle, si se trata de la 3ª o posterior iteración, ya que sino el vector de la base queda sin modificar como se vio en la ecuación. En este punto es importante destacar que la primera iteración del algoritmo, que nos permite determinar el primer endmember se opera con diferencias con respecto al resto, y por eso es a partir de la segunda iteración

cuando se comienza a generar la base y se siguen de manera estricta las ecuaciones presentadas.

El cálculo del nuevo vector de la base se lleva a cabo de manera iterativa gracias al bucle de las líneas 10-13, y que en función de la iteración en que se esté realiza más operaciones (obsérvese en las ecuaciones que a medida que la base tenga más elementos es necesario el cálculo de más constantes y la realización de más restas). Una vez se ha obtenido este vector, se procede a calcular la constante c_2, que a su vez permitirá calcular el nuevo valor de f_{acc}, en la línea 15 del algoritmo. Este valor acumulado representa la sumatoria que se ha expuesto anteriormente para obtener el valor de f.

Este valor de la acumulación se vuelve a reinicializar en la primera iteración, ya que como se dijo, esta se deja fuera de la base, que comienza a partir de que se determina el primer endmember (segunda iteración).

Entre las líneas 20 y 22 lo que se hace es normalizar el vector f, de tal forma que su menor valor absoluto sea mayor que uno y así poder realizar un redondeo en la operación de proyección, y ejecutarla en punto entero. Este procedimiento en hardware presenta unas ganancias considerables en cuanto a velocidad y utilización de recursos que se verán en el capítulo de resultados. Se trata de una operación muy simple que consiste en el desplazamiento del exponente de todos los valores del vector por igual, lo que se traduce en que todos los valores van a estar multiplicados por la misma cantidad, sin llegar a afectar al cálculo de la proyección, que si se ve ligeramente afectada por la operación de redondeo que se ejecuta en la línea 24, pero que es imprescindible para la transformación a entero. Como se explicó con anterioridad, la matriz Y también debe ser sometida al mismo proceso de normalización.

Para comprender mejor el proceso de normalización se va a mostrar un ejemplo con cadenas binarias que representan números en punto flotante con precisión simple.

$$00111111101000000000000000000000 = 1{,}25$$
$$01000000001000000000000000000000 = 2{,}5$$
$$01000000101000000000000000000000 = 5$$

El primer bit empezando por la izquierda (el MSB) es el bit se signo, a partir de ahí, los 8 siguientes bits son el exponente del número, y obsérvese como en el ejemplo, sumándole 1 al exponente cada vez se multiplica por dos el valor en punto flotante. Ejecutar esto en hardware es simple y eficiente.

Para realizar la operación de normalización se utilizan dos funciones, `fp2bin` y `bin2dec`, que simplemente se encargan de convertir primero el número en punto flotante a cadena binaria y la segunda el exponente de cadena binaria a valor entero decimal.

A partir de la línea 25 el código permanece invariable con respecto al algoritmo VCA original, se calcula el mayor valor absoluto de la proyección, se guarda el índice y se carga el nuevo endmember en la matriz `A`.

3.Diseño de arquitecturas y proceso de verificación

En el capítulo que se desarrolla a continuación, se presenta el diseño llevado a cabo para la implementación del algoritmo a bajo nivel. La arquitectura del diseño se explicará mediante una serie de diagramas de bloques que se irán adjuntando, conjuntamente con las explicaciones pertinentes.

En primer lugar se mostrará un diagrama genérico del diseño, donde se describen las funciones de los bloques principales, y posteriormente se explican en detalle las características y procesos que suceden en cada bloque.

El algoritmo se implementa de dos formas, una de ellas realizando todas las operaciones en punto flotante, incluso la proyección de la matriz `Y` sobre el vector `f`, y otra en la que se realiza dicha proyección en notación entera. A lo largo del capítulo, se resaltarán las diferencias entre cada una de estas dos versiones.

Finalmente en el capítulo se mostrará el proceso que se ha llevado a cabo para la verificación del código en hardware, comparándose los resultados con los que se obtienen en la implementación en software.

3.1. Arquitectura general

Antes de comenzar a explicar el diseño general de los algoritmos, es conveniente destacar el hecho de que durante el diseño se trabajará con números enteros y en punto flotante, tal y como se ha comentado. Por este motivo, se remite al lector a las referencias [24] y [25], donde se explican diversas operaciones de conversión y del estándar IEEE754 en relación a esta representación.

Asimismo, se han creado rutinas en Matlab para realizar la conversión, y facilitar la comparación con los resultados que se obtienen en las simulaciones con la herramienta ModelSim.

Por otra parte, en el diseño de los algoritmos se ha hecho uso de bloques incorporados en la herramienta XILINX ISE, que facilitan enormemente la labor desarrollada. Estos bloques se denominan *ip cores*, y principalmente servirán para implementar operaciones matemáticas complejas o bloques de memoria. En la referencia [26] se puede consultar más acerca de estos bloques.

Una vez comentados estos puntos, se pasa a mostrar el diseño general del algoritmo, con sus dos vertientes anteriormente mencionadas. Primero se muestra el esquema genérico de las dos implementaciones, y posteriormente se explicarán los módulos que aparecen.

En las figuras 3.1 y 3.2, se muestran los esquemas de las dos implementaciones.

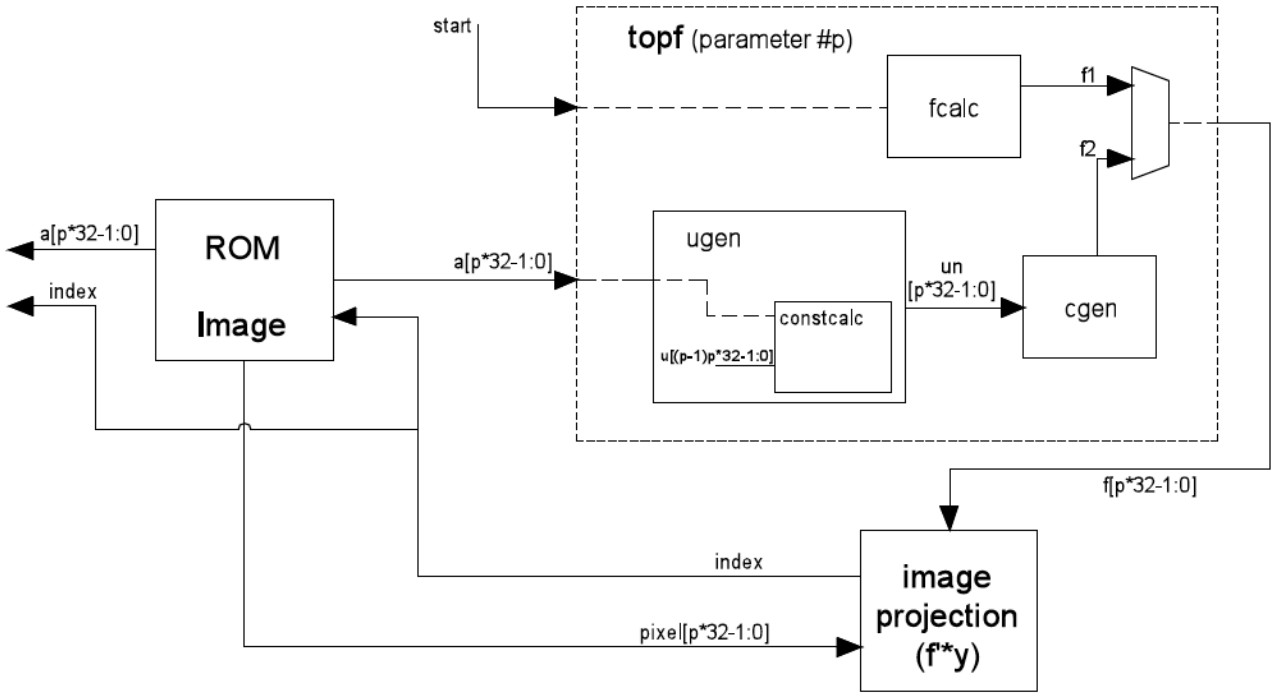


Figura 3.1. Esquema general del VCA, realizando todas las operaciones en punto flotante.

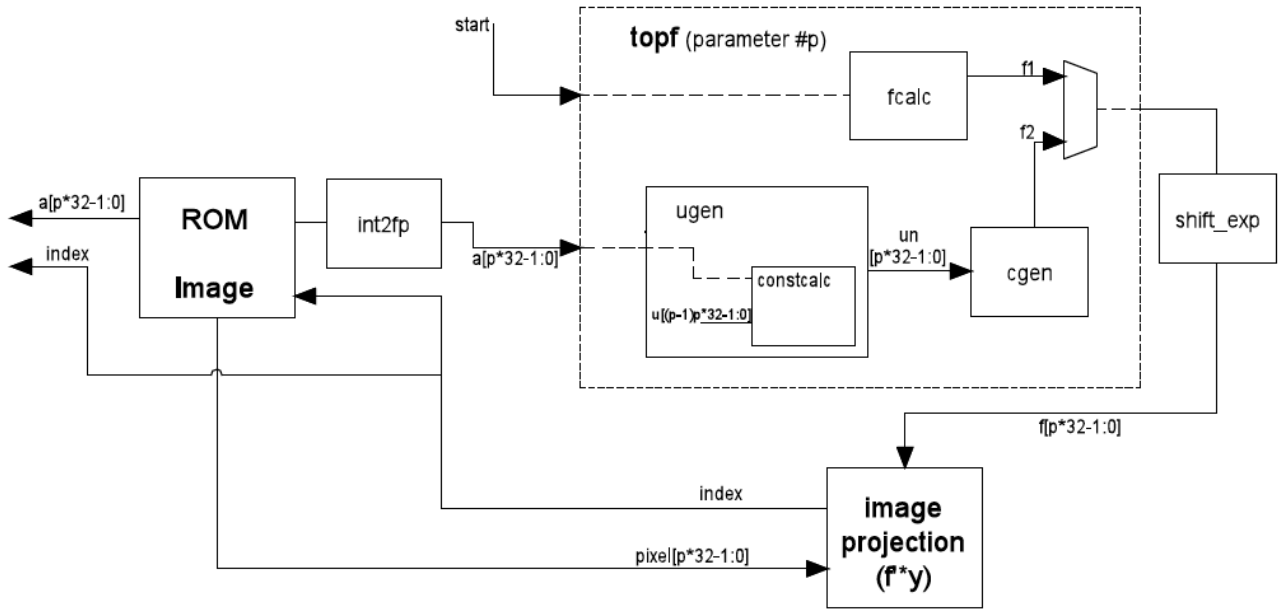


Figura 3.2. Esquema general del VCA, realizando el cálculo de f en punto flotante y la proyección de la imagen en notación entera.

En primer lugar se describe el funcionamiento genérico del algoritmo, que es compartido por las dos implementaciones, y posteriormente se destacan las particularidades de cada una de ellas.

La imagen de la cual se extraen los endmembers se almacena en una memoria ROM que es inicializada por un fichero externo que contiene los datos de la misma. El algoritmo comienza cuando lo indica una señal externa de `start`. En ese instante, el bloque interno de *topf*, *fcalc*, inicializa el vector `f`. Esta primera iteración, tal y como se comentó en la explicación teórica del método modificado, se trata de una manera diferente, y es que por un lado el vector de inicialización de A no formará parte de la base ortogonal que se crea, y por otra el vector `f` siempre se genera de la misma forma en esta primera iteración independientemente de la imagen que se procesa. Por esta razón, se genera un bloque que ejecuta esta función.

Este bloque *fcalc* sólo realiza esta función, ya que a partir de la segunda iteración, el vector `f` es calculado por el bloque *cgen*, de ahí la necesidad de un multiplexor que seleccione la señal pertinente.

Una vez se ha obtenido el vector `f` en cada iteración, se realiza la proyección de la imagen sobre este mismo, es decir, se multiplica cada píxel de la imagen por este vector. Esta operación se realiza en el bloque *image projection*, que se describe en detalle posteriormente. La salida del bloque `index` indica que píxel ha producido el mayor resultado, y por tanto este será el nuevo endmember, `a`.

Una vez obtenido `a`, este se introduce en el bloque *topf*, que se encarga de nuevamente obtener el vector `f`. Así se procede de manera sucesiva hasta completar el cálculo de todos los endmembers. Destacar que este último número se introduce en el código como un parámetro, aunque en otras aplicaciones podría venir como señal de entrada al código.

Las salidas de las dos implementaciones son las mismas, los endmembers `a` calculados en cada iteración y el índice, `index` que se obtiene en cada una de ellas. El número de endmembers a calcular, `p`, se introduce como parámetro en el código.

Los aspectos a resaltar diferentes en las dos implementaciones son los siguientes, en primer lugar los valores de la imagen en la primera implementación son almacenados en punto flotante, mientras que en la segunda se almacenan en notación entera. En este segundo caso, tal y cómo se aprecia en el código, antes de pasar el nuevo endmember seleccionado de la memoria al bloque *topf*, se debe convertir a punto flotante a través del bloque *int2fp* (que es un bloque *ip core* de la herramienta), ya que este bloque opera con esta notación.

Para realizar la proyección, se debe desplazar el exponente de todas las componentes del vector `f` de tal forma que la menor de ellas sea mayor que 1 en valor absoluto, y posteriormente volver a convertir a entero para poder realizar el producto con cada píxel de la imagen.

3.2. Arquitectura de cada bloque

En este apartado se realiza una descripción en detalle de aquellos bloques más complejos del algoritmo, que aparecen en el diagrama principal. Los bloques internos del bloque *topf* son idénticos en ambos casos, operando siempre con los números en punto flotante. Se trata de bloques con una lógica de control relativamente compleja, por lo que se ha decidido representar sus operaciones mediante diagramas de flujo.

3.2.1. Bloque *ugen*

El bloque *ugen* se encarga de obtener la nueva base de vectores `u`. Se genera un nuevo vector de la base cada vez que existe un nuevo endmember, `a`, a la entrada del algoritmo. El número de operaciones aumenta proporcionalmente al número de vectores ya calculados de la base.

3.2.1.1. Bloque *constantcalc*

El bloque presenta un bloque interno denominado *constantcalc* cuya función es la de calcular todas las constantes necesarias para el cálculo de un nuevo vector en la base. La lógica de control de este bloque se representa en la figura 3.3.

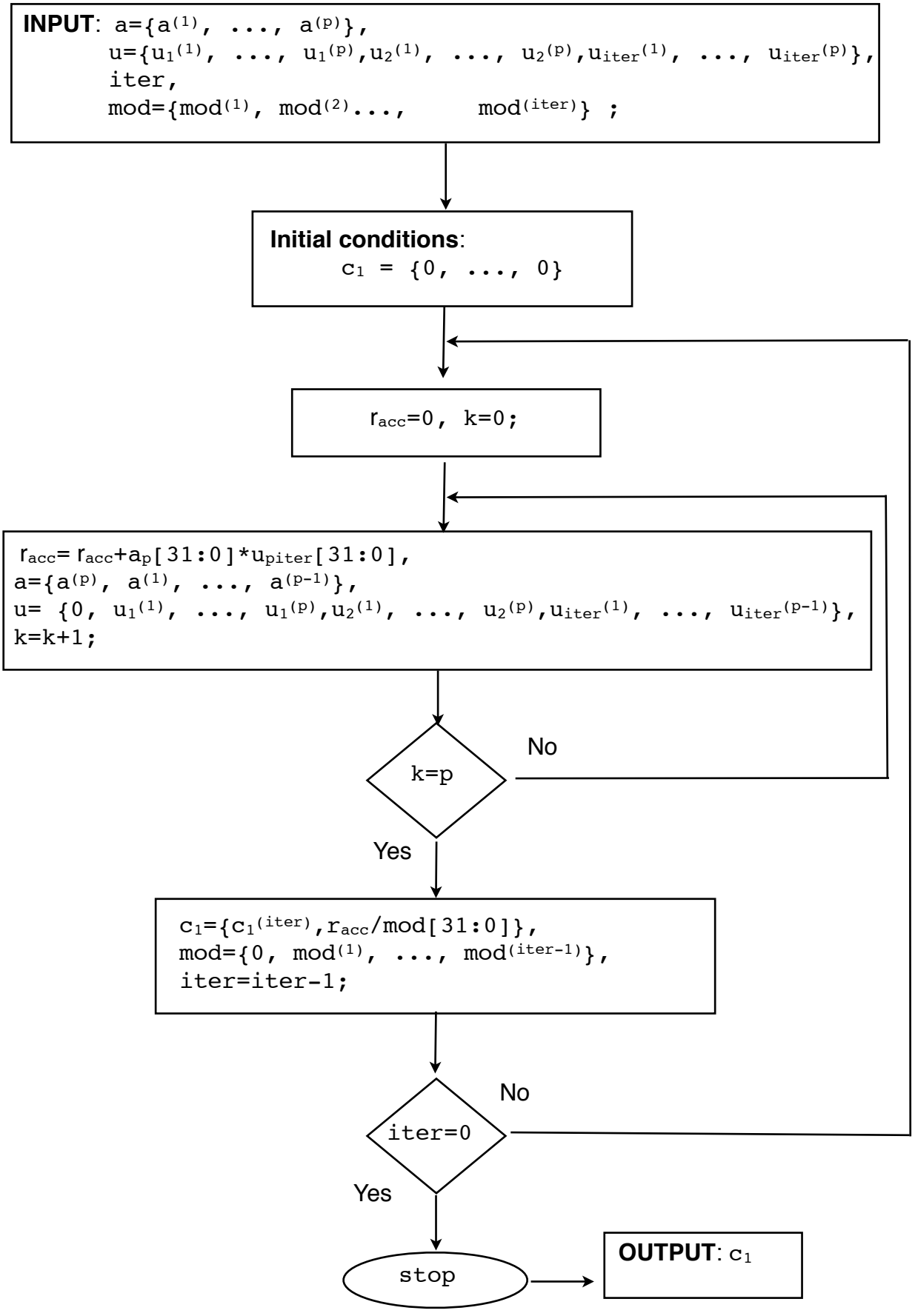


Figura 3.3. Diagrama de flujo del bloque constantcalc.

Las entradas del bloque son todos los vectores que hayan sido calculados de la base u, el nuevo endmember calculado a, la iteración en la que se encuentra el algoritmo, iter y el valor de un registro mod, que tiene almacenados los productos de los vectores de la base u que ya han sido calculados, y que suponen el denominador de las constantes que se pretenden encontrar. Estos denominadores son calculados por separado, pero siguen una lógica muy similar a la planteada.

Obsérvese que cuando no hay ningún vector en la base ortogonal calculado no es necesario entrar en este bloque, ya que no hay que calcular ninguna constante. A medida que se avanza en las iteraciones, los vectores de la base aumentan y por tanto también lo hacen las constantes a calcular. Un aspecto importante a tener en cuenta es que en hardware, los registros deben tener longitudes fijas en tiempo de síntesis, por lo que no se pueden poner en función de otro registro variable como sea en este caso iter. Lo que se hace es llenar con cero las posiciones que no se usan, lo cual no ha sido expresado en el diagrama por claridad en la representación, y porque no influye en la lógica del diseño, pero es importante tenerlo en cuenta.

La condición inicial que se fija cada vez que se ejecuta el algoritmo es establecer el vector c_1 con todos sus valores a cero, de tal forma que cada vez que se calcule un valor nuevo de la constante se desplace el vector. El registro c_1 tiene una longitud de dos valores menos que el número de endmembers a calcular, p, es decir, 32*(p-2) bits.

Además, cada vez que comienza el cálculo de una nueva constante se inicializa a cero el registro, r_{acc}, que va acumulando la suma del producto de los términos de a y de u, y el índice k que aumenta en cada iteración hasta llegar a p. Cada vez que se realiza el producto, el registro u de desplaza eliminando el valor que ha sido multiplicado, sin embargo a se rota, ya que los valores de a se volverán a usar en el cálculo de la siguiente constante (si hay que realizar el cálculo de más constantes).

En la figura 3.4 se representa un diagrama en el que se realiza una traza del bloque *constantcalc*, para aclarar el proceso de la rotación y desplazamiento de los vectores a y u, respectivamente.

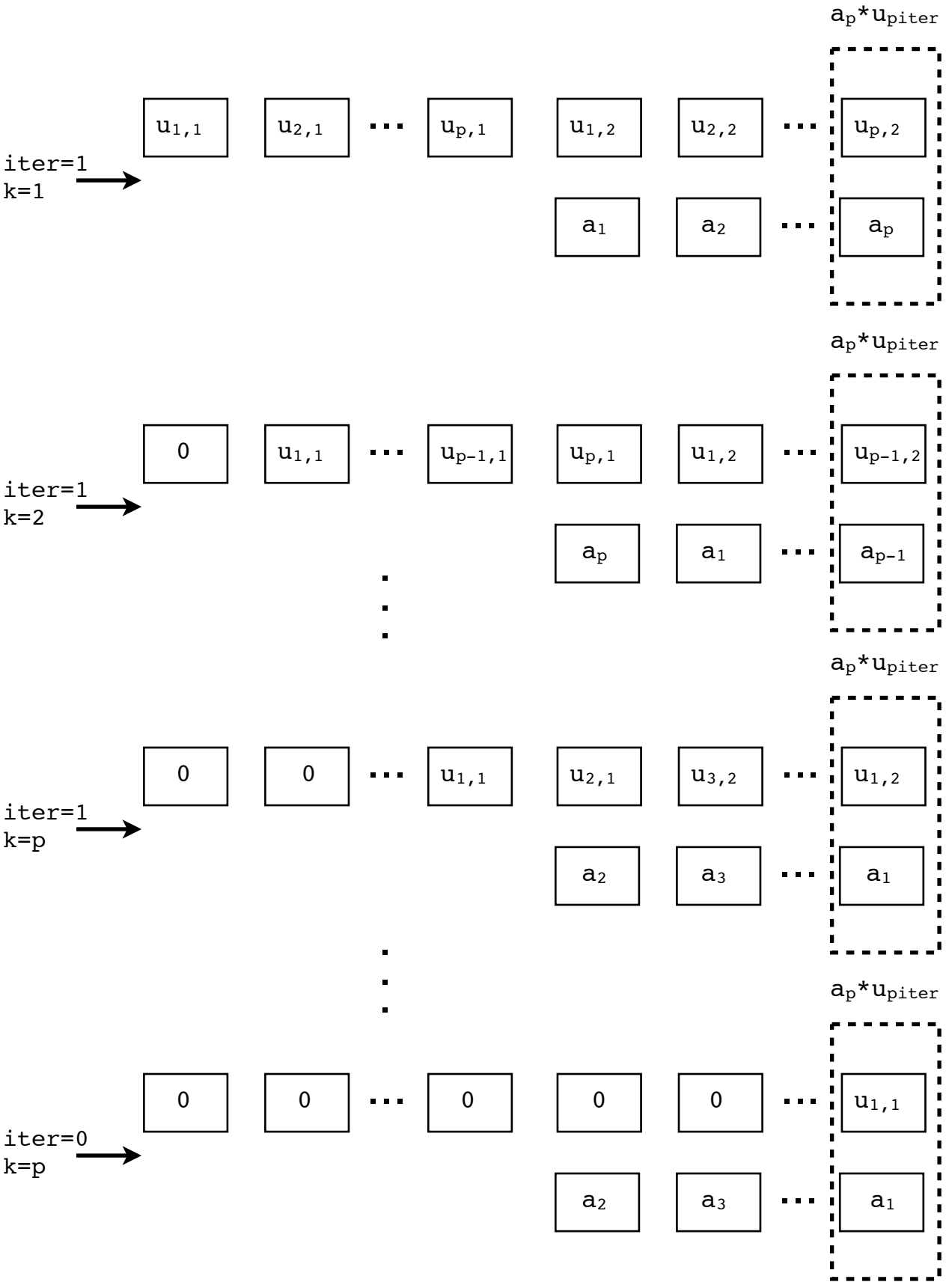


Figura 3.4. Traza del bloque constantcalc.

En la figura se observa como el vector `u` se completa con ceros a medida que se desplaza, mientras que el vector `a` es rotado, para que en las siguientes iteraciones pueda volver a enfrentar de manera adecuada al nuevo vector de la base, y que las operaciones de multiplicación se produzcan con los elementos correspondientes.

Finalmente, en el bloque *constantcalc* una vez que se completan todas las operaciones de multiplicación y `k` iguala a `p`, entonces se pasa a realizar la división y posteriormente a guardar el valor de la constante calculada. Tal y como se comentó anteriormente, el registro c_1 se encuentra lleno de ceros, y se va llenando a medida que se calculan las constantes.

Cuando el registro `iter` llega a cero, se concluye el proceso y se devuelve el registro con todas las constantes calculadas, y se procede a continuar con el bloque *ugen*, tal y como se explica en las líneas que aparecen a continuación.

Ugen

Cuando se ha completado el cálculo de las constantes c_1, se recibe una señal de aviso para que en el bloque *ugen* se comienza con el cálculo del nuevo vector de la base, nuevamente para explicar los procesos que se suceden en este bloque se recurre a un diagrama de flujo que se muestra en la figura 3.5.

El funcionamiento del bloque es muy similar al del anterior bloque, *constantcalc*. En este caso se comienza volcando en el vector u_n, el nuevo endmember obtenido `a`, al igual que se hizo en el algoritmo de modificación de VCA que se presentó anteriormente, concretamente la línea 6. A partir de ahí se le va restando a cada valor la multiplicación de la constante por el correspondiente vector de la base `u`, que había sido calculado anteriormente.

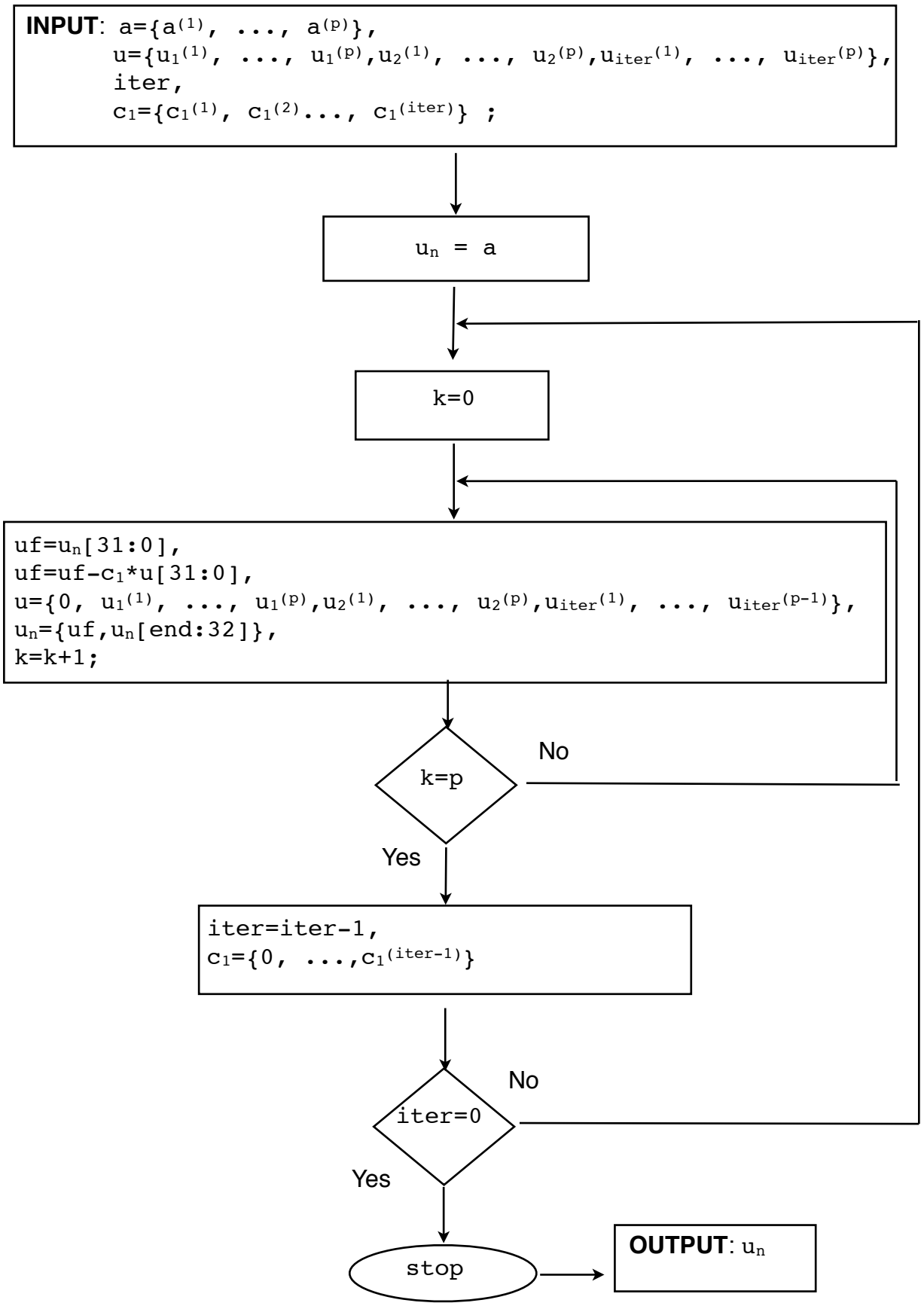


Figura 3.5. Diagrama de flujo del bloque ugen.

En este caso, para poder calcular el vector u_n, es necesario volcar cada valor sobre un registro, `uf`, operar sobre este registro y posteriormente realizar el giro del vector, tal y como se muestra en el diagrama. Se opera utilizando el mismo valor constante hasta que se cumple el giro completo, es decir `k` iguale a `p`. Cuando esto sucede, si `iter` no ha llegado a cero, se vuelve a inicializar `k` a cero, y se cambia la constante ya que se va a operar con el siguiente vector de la base.

Una vez `iter` llega a cero, se concluye la ejecución, y se entrega como salida del bloque *ugen* a *cgen* el nuevo vector de la base, u_n, que además debe ser almacenado en la propia base `u`.

3.2.2. Bloque *cgen*

Una vez se ha calculado el nuevo vector de la base `u`, se procede a calcular `f`, que será ortogonal a todos los vectores de la propia base. Para calcular este vector `f`, lo que se hace es utilizar un vector de acumulación f_{acc} en el que en cada iteración se almacenen los productos de los vectores de la base por sus respectivas constantes. Por lo tanto, el cálculo de `f` se lleva a cabo a partir de dos operaciones fundamentales.

1. Se debe calcular la constante entre el nuevo vector de la base y el vector auxiliar que se emplea para la ortogonalización, `w`. Recordando la expresión de la fórmula:

$$c_2 = \frac{\langle w, u_n \rangle}{\langle u_n, u_n \rangle} = \frac{\sum_{i=1}^{p} u_n}{\langle u_n, u_n \rangle}$$

El vector w es un vector lleno de unos, por lo que el cálculo de la constante se limita a la sumatoria del vector `un`, dividido por su producto escalar. Este procedimiento es muy similar al realizado en el bloque *constantcalc*, con la diferencia de que en ese bloque el número de constantes a calcular estaba en función de la iteración, mientras que en este bloque *cgen* siempre es una constante por iteración.

2. A continuación se calcula el vector acumulación, que se va modificando en cada iteración, tal y como se explicó en el capítulo 2, y finalmente en un mismo paso se obtiene el vector `f`. Para explicar el funcionamiento de esta segunda parte, se recurre a otro diagrama representado en la figura 3.6., en el que se asume que la constante c_2 ya ha sido calculada en el paso anterior, por claridad en la exposición de las ideas.

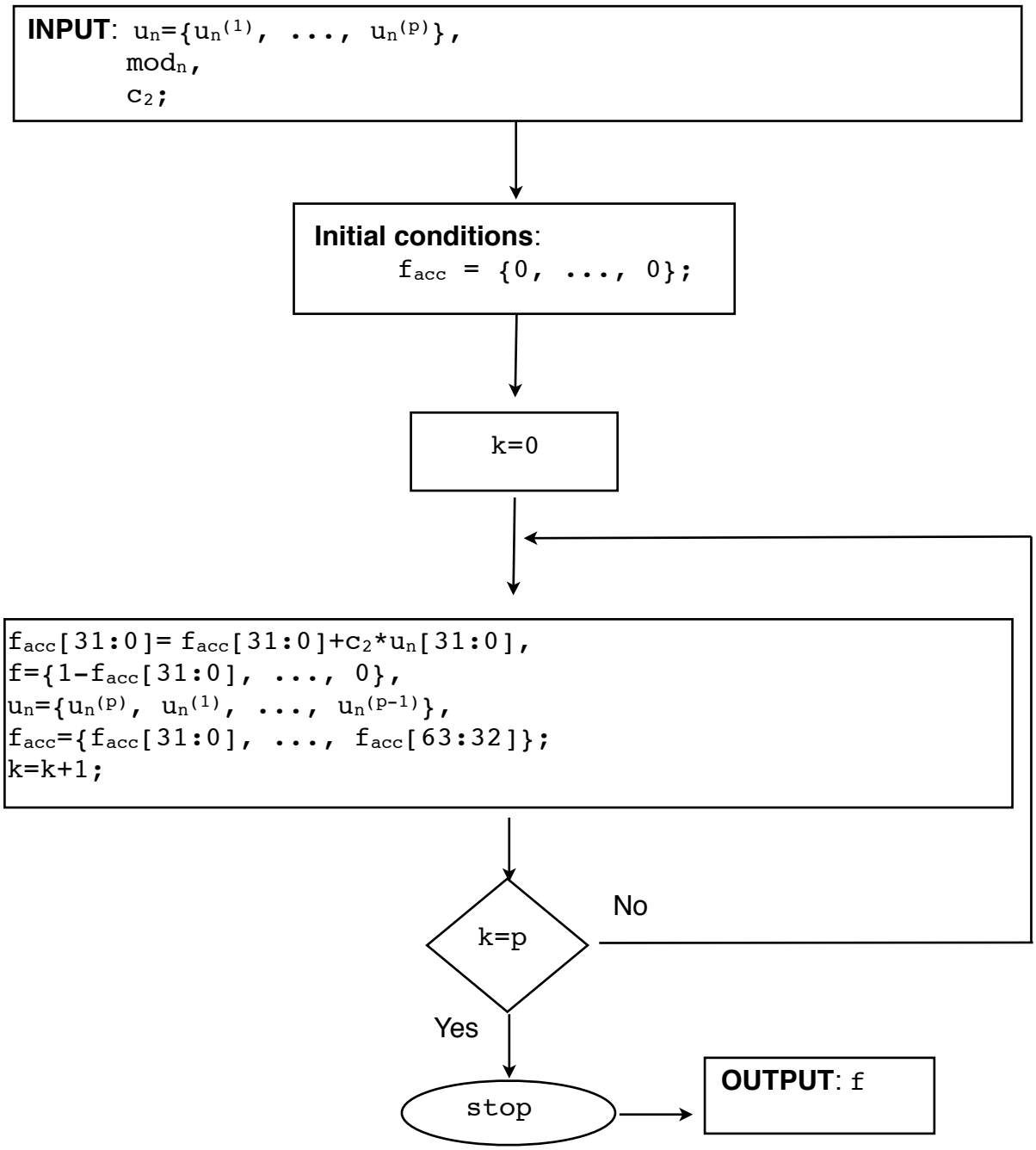


Figura 3.6. Diagrama de flujo del bloque cgen.

Se parte del vector u_n, calculado en el bloque *ugen*, el producto escalar por si mismo mod_n, también calculado en ese bloque, y la constante c_2. La condición inicial de operación es la inicialización a cero del acumulador f_{acc}, que sólo se debe realizar en el primer caso, ya que a partir de ahí debe almacenar el valor para las siguientes iteraciones.

Posteriormente se comienza a iterar, nuevamente p veces, calculando en primer lugar cada uno de los valores f_{acc} e inmediatamente utilizando este valor, el elemento correspondiente del vector f. Se rota el vector u_n, y el vector f_{acc}.

Se concluye el proceso, pasando a la salida el nuevo vector f calculado.

3.2.3. Bloque *image projection*

En el bloque image projection se realiza la proyección de la imagen sobre el vector f. Se trata por tanto de un bloque crítico en cuanto a la ejecución del algoritmo ya que en él se consume la mayor parte del tiempo de ejecución, siendo este dependiente del número de píxeles que tenga la imagen.

Destacar que para una mayor eficiencia de los recursos y rapidez en la ejecución, las operaciones se producen tipo pipeline, de tal forma que los píxeles van entrando en el proceso de manera sucesiva en cada ciclo de reloj, y por tanto el número de ciclos de operación del bloque se extiende al número de píxeles de que conste la imagen. La arquitectura de este se presenta en la figura 3.7.

El bloque recibe el píxel de la imagen (recibe uno en cada ciclo de reloj) y el vector f. A partir de ahí, como se puede observar, este consta de tres partes fundamentales. En primer lugar se realiza la multiplicación de cada uno de los elementos del píxel, por cada uno de los elementos del vector f. Seguidamente se construye un árbol de sumas, para que se pueda realizar el proceso en pipeline, siendo el número de bloques que lo constituyen en cada etapa potencia de 2. En este caso se ha implementado una primera etapa donde se suman 8 elementos, con cuatro bloques de suma, después 4 elementos con dos bloques de suma y finalmente se suman dos elementos, en la última etapa con

un sólo bloque se suma. Obsérvese que p, no tiene por qué coincidir con 8 que es el número máximo de elementos que se suman y por tanto determina el máximo de endmembers a calcular, en caso de que p sea menor que este número, el resto se rellena con ceros, tal y como se muestra en el diagrama.

Por último se realiza la comparación, que en la primera iteración simplemente toma el primer valor de la multiplicación, y a partir de ahí lo compara con las que se vayan sucediendo, almacenando aquellas que superen al valor actual y el índice de la misma, que será la dirección que se emplee posteriormente para obtener el endmember de la memoria que almacena la imagen.

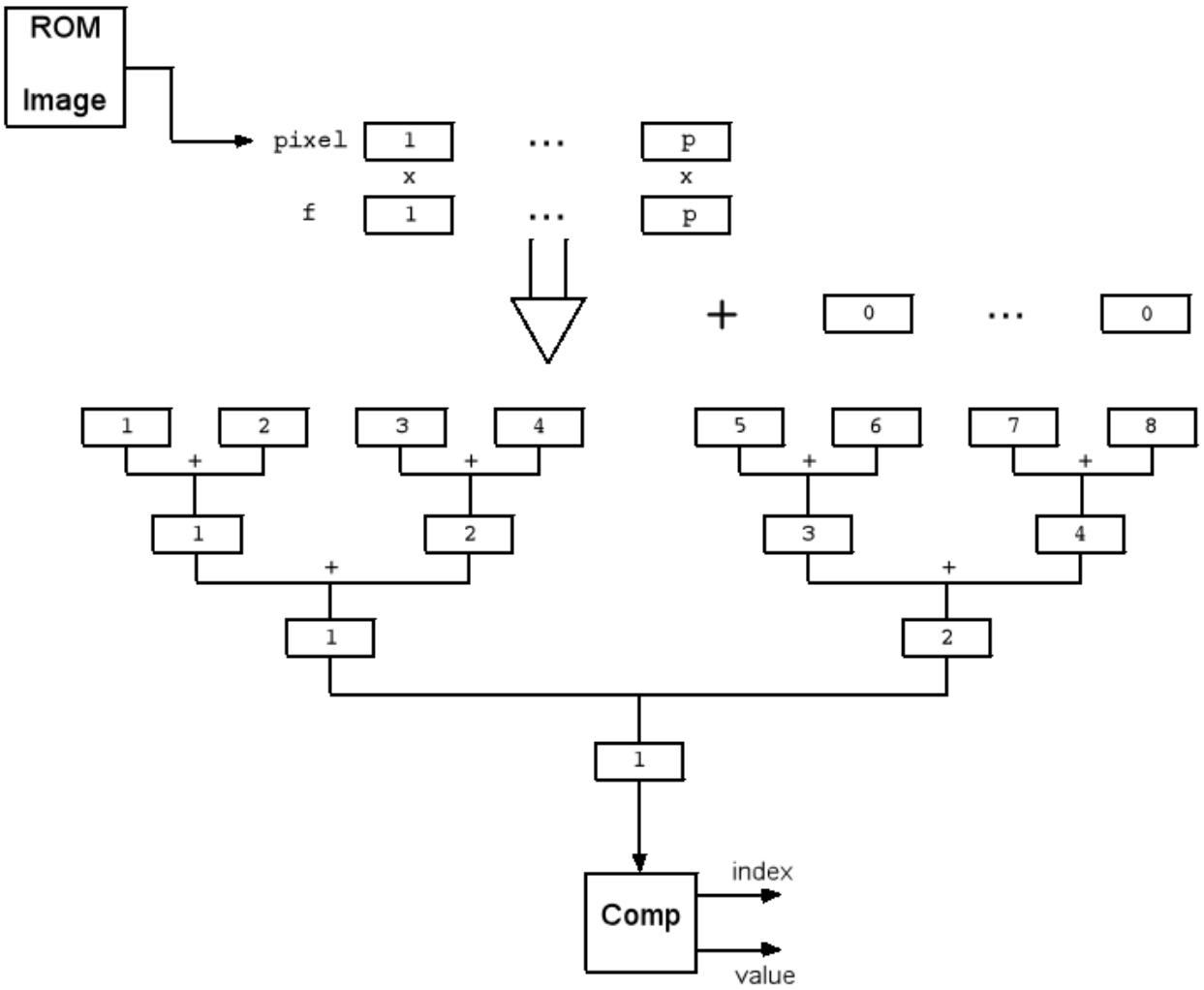


Figura 3.7. Arquitectura del bloque image projection.

El bloque entrega a la salida el valor que ha resultado ser el mayor de todos los productos, y el índice del mismo.

Un aspecto a resaltar en la comparación, es que bien se realice con números en punto flotante o con números en notación entera, esta se lleva a cabo de la misma forma, se elimina el bit de signo (para tratar el número en valor absoluto) y se busca el primer 1 que aparezca diferente en los dos números, recorriéndolo de izquierda a derecha. Aquel que tenga ese 1 más a la izquierda será el mayor.

3.2.4. Bloque *shift_exp*

Este bloque es exclusivo de la segunda implementación, realizando la proyección de la imagen con los números en notación entera. El objetivo de este bloque es convertir los elementos del vector f, de punto flotante a entero, ejecutándose en dos pasos:

1. Se desplaza el exponente de tal forma que el menor exponente sea mayor o igual a 127. Recuérdese que los números en punto flotante con exponentes mayores o iguales a 127 son números reales mayores o iguales a 1.
2. Una vez se ha hecho esta modificación se procede a realizar la conversión de punto flotante a entero, conversión que lleva implícito el truncamiento de los decimales, y que se lleva a cabo a partir de uno de los bloques de la herramienta XILINX ISE, *ip cores*.

Para ejecutar el paso 1, simplemente se ha de encontrar el menor número de los elementos de `f`, para lo cual se procede de una forma similar a como se realizó el cálculo de la máxima proyección en el bloque *image projection*. Se inicializa un registro `min` a cero, y se va comparando este registro con cada uno de los exponentes de los elementos de `f`, que están representados en punto flotante de precisión simple y por tanto las señales del exponente serían `f[30:23]`. Una vez se calcula el valor del mínimo de los exponentes se le resta a 127 este valor, y se aplica esta diferencia a todos los números. Esto produce que todos los valores de `f` sean mayores que la unidad, y por tanto el truncamiento no afecta de manera tan severa los resultados.

3.3. Verificación

Una parte fundamental en el diseño de cualquier circuito consiste en su verificación. Mediante la verificación es posible detectar los errores que se hayan podido cometer durante las distintas fases del diseño, y de esta manera poder corregirlos antes de que el circuito sea fabricado. Otra finalidad importante de la fase de verificación es poder comprobar si el diseño se comporta de la forma esperada y si se ajusta de esta manera a las especificaciones iniciales.

Se debe diferenciar entre la verificación funcional del diseño, que es justamente la que se trata en este apartado, de la verificación física, que se lleva a cabo una vez que el diseño ha sido fabricado y que tiene por objetivo detectar errores que se hayan podido producir durante el proceso de fabricación del circuito.

Muchas de las operaciones que se llevan a cabo se realizan en punto flotante, notación que no es intuitiva a la hora de reconocer el valor numérico que se trata. Por tanto, la tarea de corregir los errores en el código es compleja, y requiere de un trabajo adicional, que se ha llevado a cabo usando la herramienta Matlab, cuyo objetivo consiste en imprimir en diferentes ficheros los resultados de las distintas variables del sistema, representándose los valores en binario y en hexadecimal. De esta forma la comparativa se puede llevar a cabo y realizar una depuración óptima del código.

Para poder llevar a cabo la tarea mencionada es fundamental el desarrollo de ciertas rutinas en Matlab que permitan la conversión de valores numéricos y la impresión de los mismos en ficheros. Destacar que en el entorno Matlab, las cadenas binarias y hexadecimales son tratadas como cadenas de caracteres.

Una vez que las rutinas en Matlab han sido creadas, se procede a realizar las simulaciones del código RTL. Para ello se emplea la herramienta ModelSim, donde se pueden observar todas las señales que forman parte del diseño sometido a test.

El proceso de verificación ha estado constituido por dos fases principales.

Una primera fase se encarga de verificar el correcto funcionamiento de los bloques individuales que componen el diseño, en nuestro caso son topf, msc y la ROM. El primero de los bloques es el más complejo y de mayor lógica, englobando a su vez otros bloques. Se construye un testbench que se encarga de proporcionar los endmembers y se comprueba que a la salida el vector f calculado es el correcto, a través de los ficheros de datos generados.

Se procede de igual manera con la verificación de *msc*, se genera un *testbench* que se encargue de proporcionar de manera consecutiva los píxeles de la imagen, y que a su vez también disponga del vector `f`, por lo que en la proyección se puede comprobar si los índices que salen como ganadores en todas las proyecciones son los correctos.

La segunda fase se encarga de ensamblar todo el código, comprobando el correcto funcionamiento de todos los bloques juntos. En este caso, el *testbench* que se debe generar es muy simple y consiste en que simplemente una señal indique el comienzo de todas las operaciones.

La herramienta ModelSim también permite generar ficheros con los resultados, lo que ha facilitado mucho la tarea, ya que simplemente esta ha consistido en generar dichos ficheros y compararlos con los que ya se disponía en Matlab, para ir controlando cada uno de los pasos del código.

4.Resultados

En el capítulo anterior se presentaron las arquitecturas de las dos implementaciones, la lógica de algunos de los bloques principales a través de diagramas de flujo, y finalmente se expusieron los procesos de verificación que se llevaron a cabo para la correcta implementación del algoritmo a nivel RTL.

En este capítulo se presentarán los resultados que se obtienen, por un lado en la implementación que se ha hecho en el software Matlab, utilizando el algoritmo descrito en el capítulo 2 y estableciendo una comparación con los resultados del algoritmo VCA original, y por otro lado los resultados de las dos implementaciones llevadas a cabo en hardware, y sintetizadas en una FPGA Virtex 5.

Para mostrar los resultados en Matlab, se utilizarán las métricas de comparación presentadas también en el capítulo 2, y se hará uso del entorno de simulación desarrollado por los creadores del VCA, que también será presentado.

4.1. Demo VCA

Los desarrolladores del algoritmo VCA, han creado conjuntamente con este, un entorno de test. Se trata de una demo que engloba tres posibles análisis del algoritmo VCA. Antes de ver estos tres test, vamos a ver el concepto de imagen artificial, que se emplea en los mismos, tal y como se verá.

La imagen artificial, es una herramienta muy útil a la hora de probar los algoritmos de extracción de endmembers, ya que consiste en crear una imagen a partir de una serie de firmas espectrales puras (endmembers reales) y mezclar estos píxeles, para componer la imagen. La mezcla se puede realizar de forma manual, metiendo los datos de las abundancias en una matriz, o a partir de funciones estadísticas que hagan el reparto de las abundancias. En estas imágenes la relación señal ruido también se puede ajustar, añadiendo a cada píxel una componente de ruido gaussiano.

Por tanto la demo consta de un primer test donde las abundancias son introducidas de forma manual por el usuario con la única posibilidad de obtener 3 endmembers.

En segundo caso, también se construye una imagen artificial, en este caso a partir del número de firmas espectrales que se desee (destacar que las firmas espectrales se obtienen de la librería USGS [18]), y las abundancias se generan de forma aleatoria a partir de una función de dirichlet. En estos dos casos en que se crean imágenes artificiales, estas presentan un tamaño de 36x36 y 224 bandas espectrales, esta última dimensión se debe a que las firmas espectrales de las librerías tienen 224 bandas. El tamaño de la imagen, no obstante, puede ser modificado al que se desee.

El último caso consiste en detectar los endmembers de una imagen real, que además haya sido profundamente estudiada, tal como la imagen Cuprite, (figura 4.1) donde se muestra una de las bandas de frecuencia de dicha imagen que tiene un tamaño de 250x191 píxeles, y 188 bandas espectrales. De esta imagen se conocen muy bien los resultados de los materiales que existen en la escena y por tanto se puede nuevamente comprobar si el método funciona correctamente.

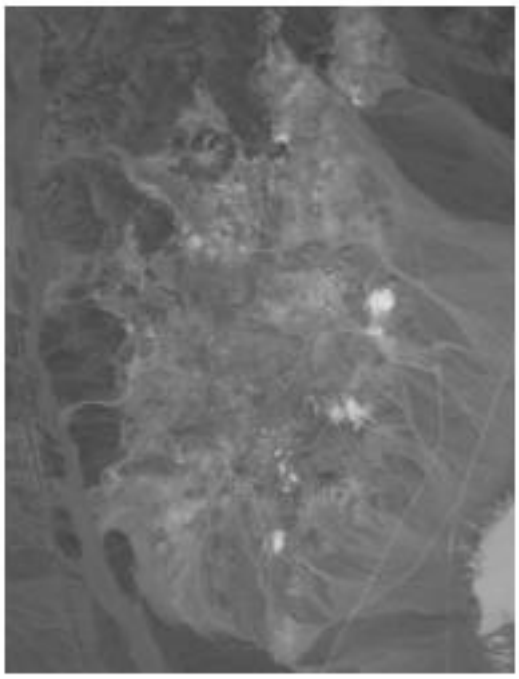


Figura 4.1. Imagen Cuprite (250x191 píxeles y 188 bandas espectrales).

En la figura 4.2 se puede observar el resultado del test 2, utilizando 4 firmas espectrales para construir la imagen artificial. Esto supone que la imagen tiene 4 endmembers que se han de determinar con el algoritmo VCA.

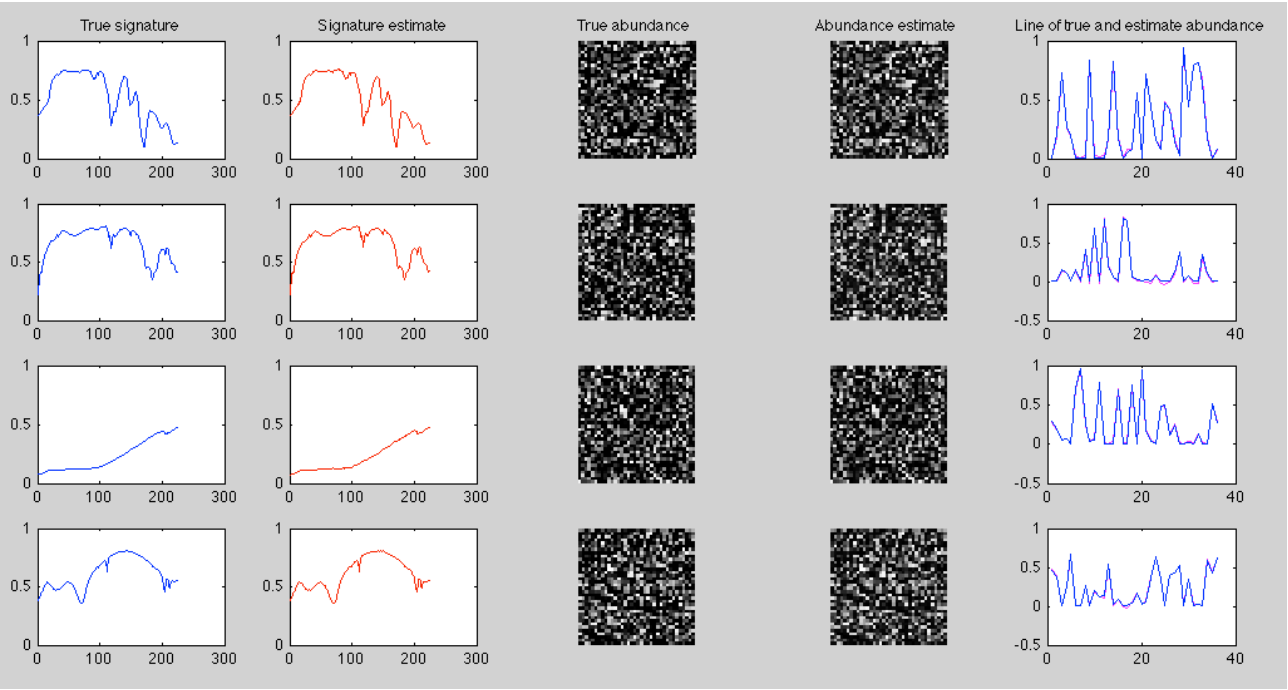


Figura 4.2. Resultados del test 2 con 4 firmas espectrales.

En la imagen se observa, en la parte izquierda la firma espectral original, de la que se parte para construir la imagen, e inmediatamente después la firma obtenida a partir del método VCA. Posteriormente se muestran las imágenes de abundancia originales (que se obtuvieron de manera aleatoria a partir de la función de Dirchlet) y la abundancias estimadas para cada endmember. Se trata de imágenes complejas de analizar visualmente, aunque si observamos detalladamente los patrones en ambos lados, vemos que son idénticos, variando ligeramente la intensidad de las dos imágenes. Por último se muestran unos gráficos donde se han superpuesto las abundancias originales y las obtenidas de una línea concreta de la imagen (la primera línea de la imagen).

Los resultados visuales mostrados en la figura 4.2 fueron obtenidos utilizando el algoritmo VCA modificado que se presentó en el capítulo 2. Tal y como se ha comentado, el análisis visual es muy importante para comprobar que el método funciona correctamente. Sin embargo debemos apoyarnos en figuras de mérito para contrastar los resultados. Estos resultados se presentan en el siguiente apartado.

4.2. Resultados en software con imágenes artificiales

En este apartado se construye una tabla con los resultados numéricos que se obtienen con el algoritmo VCA original y modificado. Se hace uso de la demo 2, que se explicó anteriormente, y como figura de mérito se emplea una ponderación, en concreto el valor eficaz (rms), de todas las diferencias entre los endmembers originales que se emplearon para obtener la imagen artificial, y los endmembers estimados tanto por uno como por el otro algoritmo. Representándose finalmente la resta entre los valores que generan un método (VCA original) y el otro (VCA modificado). El valor que se emplea en la comparativa es:

$$\theta_i \equiv (arc\cos\frac{\left\langle m_i, \hat{m}_i \right\rangle}{\left\| m_i \right\| \left\| \hat{m}_i \right\|})$$

$$\theta_{RMS} = \left(E\left[\left\| \vec{\theta} \right\|^2 \right] \right)^{1/2}$$

donde $\vec{\theta}$ es el vector que contiene a todos los ángulos que existen entre los endmembers originales y los estimados, siendo calculados estos como se expresa en la primera fórmula.

Los resultados que se obtienen con ambos métodos son dependientes tanto del parámetro p, número de endmembers a calcular, como del SNR, relación señal ruido, por lo que se construye una tabla en la que por un lado varíe el número de endmembers, y por otro varíe el SNR. Se ejecuta el algoritmo 10 veces para cada una de las casillas de la tabla, y se obtienen los resultados para el VCA original y el modificado, estos se muestran en las tablas 4.1 y 4.2.

SNR →

p ↓	10 dB	20 dB	30 dB	40 dB	50 dB
2	9,24º	0,9608º	0,2838º	0,0876º	0,0215º
3	6,3432º	1,1433º	0,3186º	0,0899º	0,0308º
4	6,5027º	3,534º	0,3713º	0,1143º	0,0459º
5	4,5106º	2,8869º	0,4747º	0,1565º	0,0718º
6	6,6507º	4,0686º	0,4854º	0,6037º	0,0984º
7	6,4881º	1,9999º	0,8873º	0,1892º	0,2536º
8	6,3709º	1,9645º	0,5815º	0,2732º	0,1244º
9	8,7916º	2,2782º	0,7226º	0,4342º	0,2065º
10	13,045º	3,7923º	0,9757º	0,3641º	0,2183º

Tabla 4.1. Media de los valores eficaces ($\overline{\theta}_{RMS}$) en el algoritmo VCA original.

SNR →

p ↓	10 dB	20 dB	30 dB	40 dB	50 dB
2	21,3º	0,8944º	0,2521º	0,1694º	0,1521º
3	7,3366º	1,1302º	0,3307º	0,1019º	0,0302º
4	6,4841º	4,0594º	0,4233º	0,1239º	0,0568º
5	4,6569º	2,0124º	0,432º	0,2921º	0,0654º
6	5,5028º	4,5647º	0,5503º	0,1795º	0,0861º
7	6,0469º	3,19º	0,4912º	0,2367º	0,1345º
8	7,0814º	2,5667º	0,7101º	0,2135º	0,1085º
9	9,283º	3,3616º	0,8023º	0,3237º	0,2103º
10	10,774º	3,7698º	0,9019º	0,3745º	0,275º

Tabla 4.2. Media de los valores eficaces ($\overline{\theta}'_{RMS}$) en el algoritmo VCA modificado.

Una característica importante a resaltar de las dos implementaciones del VCA, es que a medida que aumenta el número de endmembers, la tendencia es que el error sea mayor, y a medida que aumenta el SNR disminuye el error.

Para obtener una idea de la diferencia que existe entre ambos métodos, se van a restar ambas tablas, de tal forma que si los números que aparecen en cada celda individual son negativos, significa que el VCA original a calculado mejor el resultado, mientras que si son positivos significa que el VCA modificado lo ha calculado mejor. Los resultados se pueden ver en la tabla 4.3.

SNR →

p ↓

	10 dB	20 dB	30 dB	40 dB	50 dB
2	-12,06º	0,0664º	0,0317º	-0,0818º	-0,1306º
3	-0,9934º	0,0131º	-0,0121º	-0,012º	0,0006º
4	0,0186º	-0,5254º	-0,052º	-0,0096º	-0,0109º
5	-0,1463º	0,8745º	0,0427º	-0,1356º	0,0064º
6	1,1479º	-0,4961º	-0,0649º	0,4242º	0,0123º
7	0,4412º	-1,1901º	0,3961º	-0,0475º	0,1191º
8	-0,7105º	-0,6022º	-0,1286º	0,0597º	0,0159º
9	-0,4914º	-1,0834º	-0,0797º	0,1105º	-0,0038º
10	2,271º	0,0225º	0,0738º	-0,0104º	-0,0567º
Sumatoria	-10,5229º	-2,9207º	0,207º	0,2975º	-0,0477º

Tabla 4.3. Resta de los valores eficaces de los ángulos ($\bar{\theta}_{RMS} - \bar{\theta}'_{RMS}$) obtenidos en el algoritmo VCA original y modificado.

Como se puede apreciar los valores de cada celda individual son muy pequeños, como norma general, aunque existe algún valor que se va fuera de rango, sobre todo en el caso de un SNR bajo como en la primera celda. El hecho de que los números sean muy pequeños implica que los resultados de ambos métodos son muy parecidos.

Por otra parte se ha calculado una sumatoria de todas las columnas que nos indica de manera general qué algoritmo se equivoca más. En caso de que el valor de la sumatoria sea negativo el algoritmo VCA modificado comete más errores, mientras que si es positivo este comete menos. Es importante destacar que si alguna de las celdas

presenta un valor muy grande, esto va a producir una alteración de los resultados, como es el caso de la primera columna, no obstante, este método nos sirve de manera orientativa.

4.3. Resultados en software con la imagen Cuprite

Para obtener los resultados que a continuación se presentan se ha utilizado la demo 3 introduciendo algunas modificaciones fundamentales. Se trata de un panorama real, lo que supone que el número de endmembers no es conocido a priori. Existen algoritmos que nos permiten estimar este parámetro, sin embargo en la aplicación que se lleva a cabo se realiza otro procedimiento.

Se parte de 5 endmembers que son conocidos en la escena, estos son los materiales Alunite, Budinggtonite, Calcit, Kaolinite y Muscovite. Se calculan 19 endmembers de la imagen utilizando los dos métodos, y de esos 19 se calculan todos ángulos espectrales con los 5 materiales mencionados, el que menor ángulo presenta, se asigna como el material en cuestión y dicho material se retira del test, para que en la siguiente iteración no se pueda repetir dicho material.

Las pruebas se realizan 10 veces y se obtiene la media de los resultados. Se presenta como figura de mérito la media del valor absoluto de la diferencia entre los ángulos calculados entre uno y otro método. Los resultados se muestran en la tabla 4.4.

Material	$\lvert\varphi_{VCA} - \varphi_{VCA\,\mathrm{mod}}\rvert^{\circ}$
Alunite	0,7765
Budinggtonite	0,3067
Calcit	0,7498
Kaolinite	2,0665
Muscovite	0,7339

Tabla 4.4. Representación de los valores medios $\lvert\varphi_{VCA} - \varphi_{VCA\,\mathrm{mod}}\rvert^{\circ}$ expresado en grados, obtenidos en el algoritmo VCA original y modificado, realizado con 10 iteraciones.

De los resultados se destaca el valor más elevado que el resto en el caso de la kaolinite, sin embargo, en general son valores muy bajos, lo que prueba nuevamente que los métodos obtienen endmembers muy parecidos. A continuación se muestran las gráficas con los materiales, los endmembers que detecta un método y los que detecta el otro, en las figuras 4.3, 4.4, 4.5, 4.6, y 4.7.

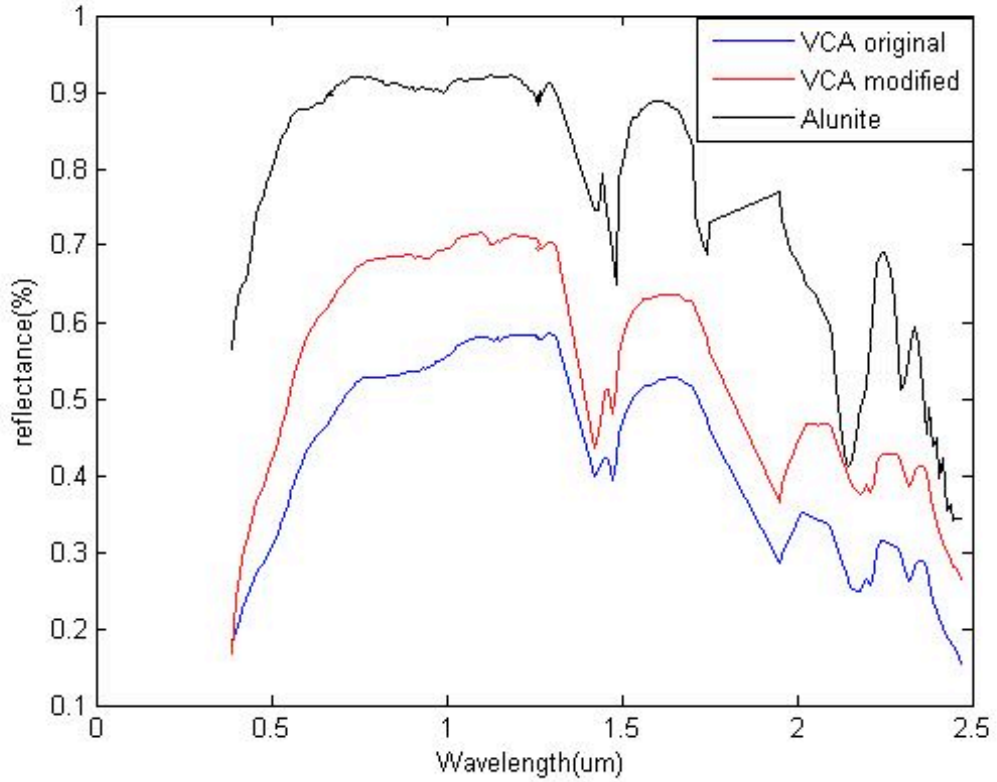


Figura 4.3. Representación gráfica del espectro Alunite, y los endmembers de cada método, con mayor similitud.

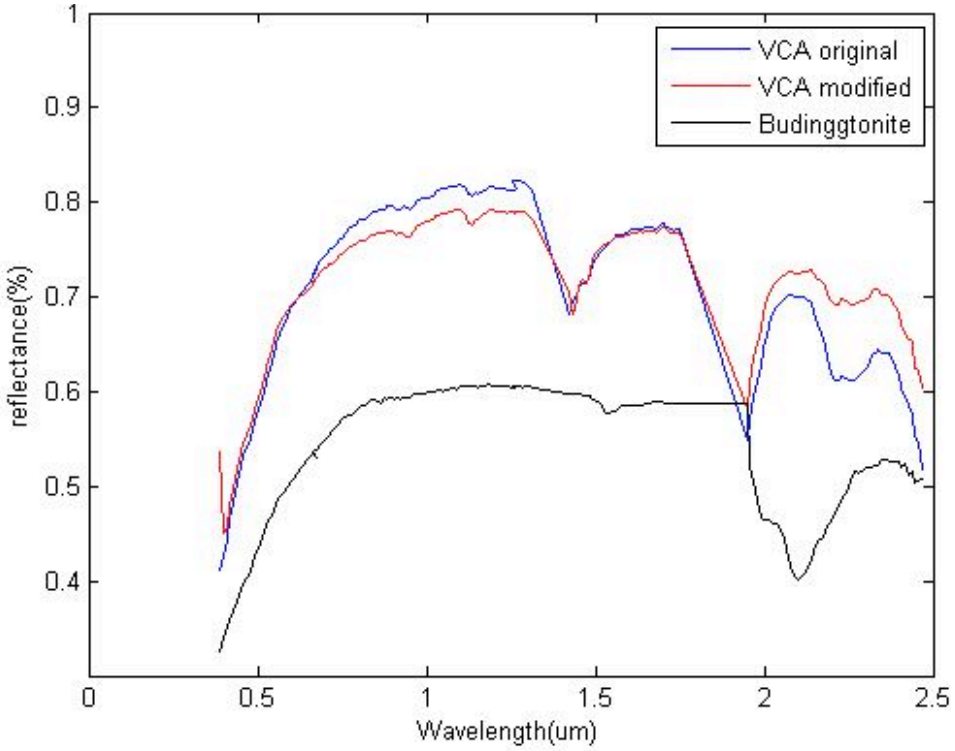


Figura 4.4. Representación gráfica del espectro Budinggtonite, y los endmembers de cada método, con mayor similitud.

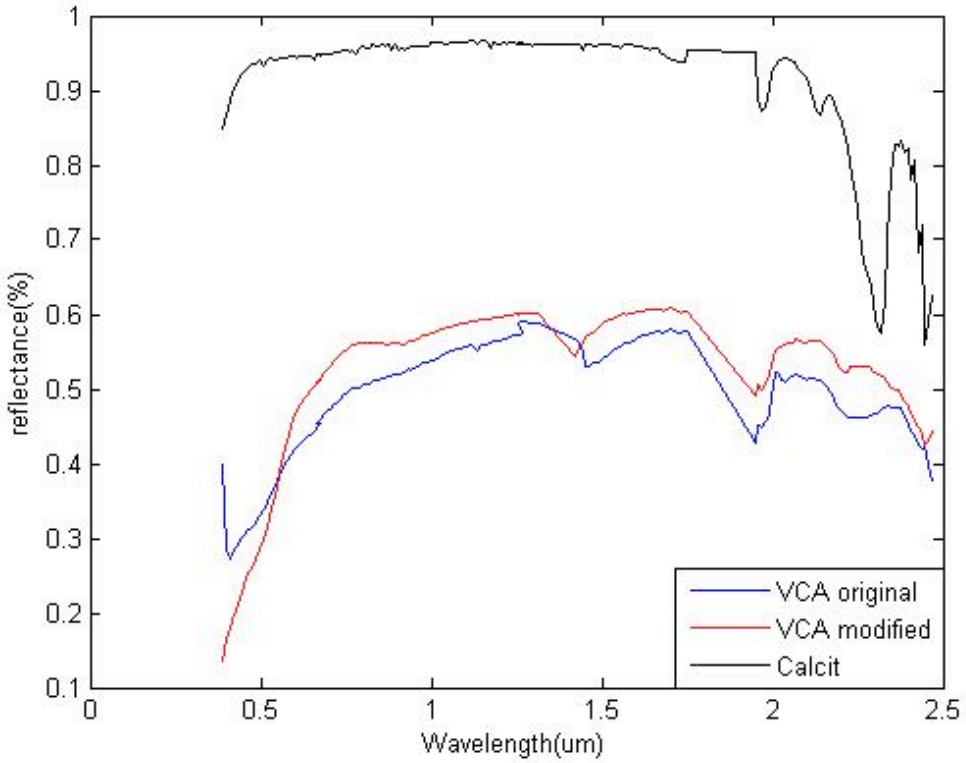


Figura 4.5. Representación gráfica del espectro Calcit, y los endmembers de cada método, con mayor similitud.

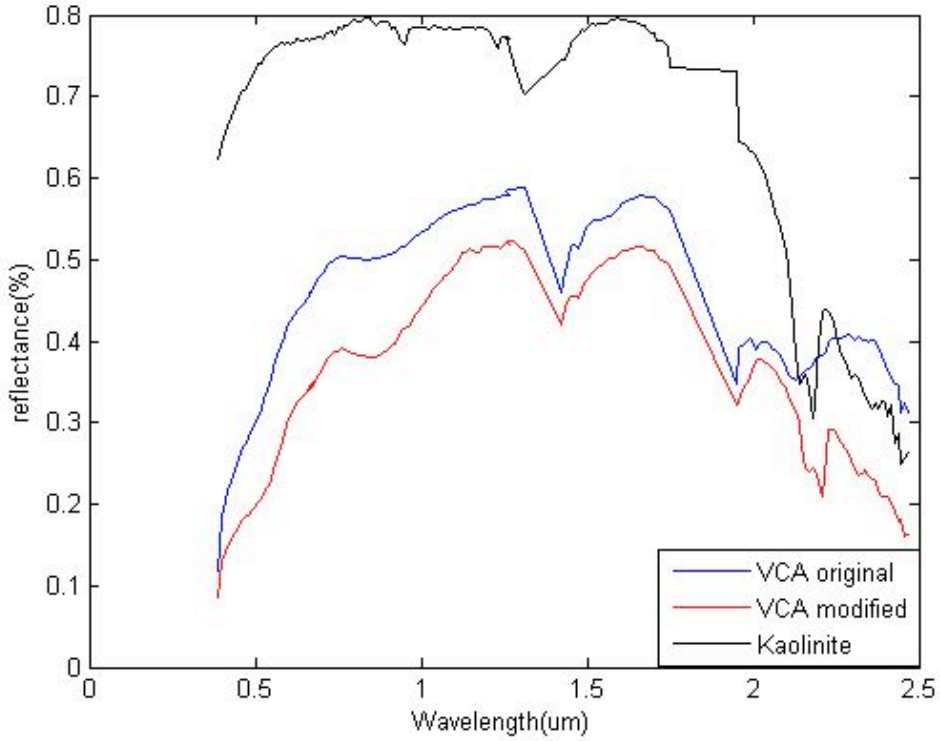


Figura 4.6. Representación gráfica del espectro Kaolinite, y los endmembers de cada método, con mayor similitud.

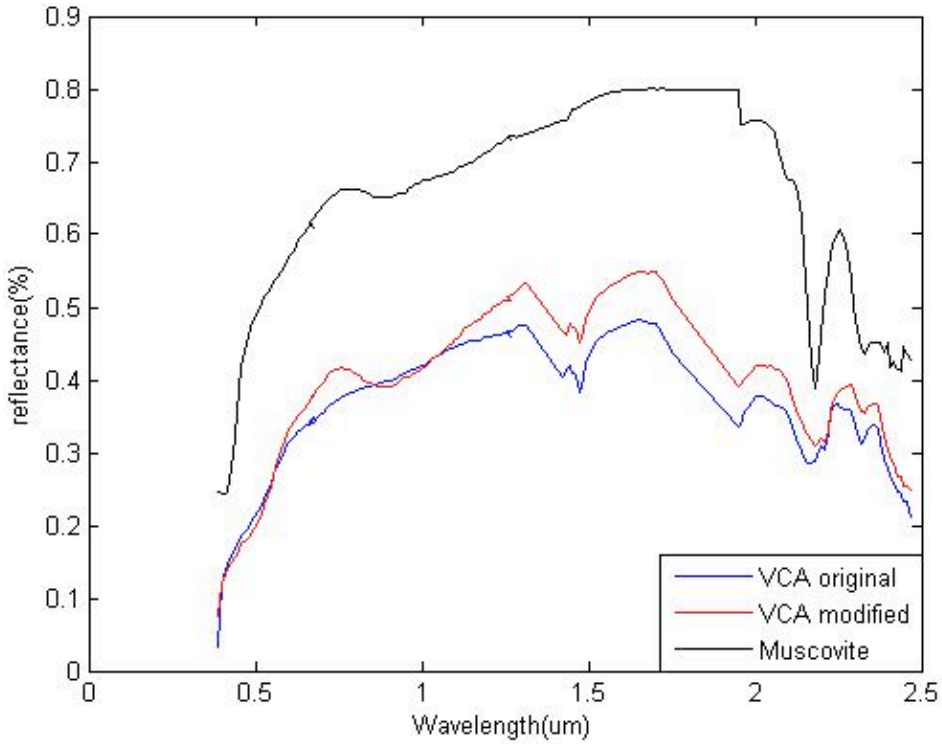


Figura 4.7. Representación gráfica del espectro Muscovite, y los endmembers de cada método, con mayor similitud.

Como se observa en todos los casos la similitud de las firmas espectrales es notoria, sobre todo las firmas azules y rojas que representan los endmembers obtenidos por el VCA original y el modificado. Esto explica la razón por la que la diferencia de los ángulos espectrales con respecto a la firma del material sea tan baja.

4.4. Resultados de la implementación hardware

Los resultados en hardware se determinan a partir de la síntesis del código descrito en lenguaje verilog, sobre una FPGA basada en una Virtex 5. Para ello se hace uso de la herramienta Xilinx ISE, mencionada en el primer capítulo.

Se desea obtener una comparativa entre las dos implementaciones, es decir, la implementación que realiza todas las operaciones en punto flotante y la implementación que realiza la proyección de la imagen en notación entera. Para esta comparativa se presentarán por un lado los resultados de ocupación sobre la FPGA, relativos al número registros, LUTs, bloques de memoria, y bloques dedicados a la multiplicación (DSP48E). Por otro lado, de la propia síntesis también se quiere conocer el valor de frecuencia máxima de uso que proporciona la herramienta. Con este valor, y con la latencia del algoritmo, que se obtiene utilizando un contador que se inicializa cuando comienza la ejecución del algoritmo, y se detiene a la conclusión, se puede conocer el tiempo total de ejecución para unas condiciones determinadas.

En este caso se trabaja con las imágenes artificiales que tienen un tamaño de 36x36 píxeles y 224 bandas espectrales. Sin embargo, en el algoritmo se introduce la proyección de la imagen que en vez de presentar las 224 bandas espectrales, presenta una tercera dimensión igual al número de endmembers que se desea calcular. En este caso el parámetro p, número de endmembers, se ha fijado en 5.

Los resultados de ocupación y de restricciones temporales de las dos implementaciones se pueden observar en las tablas 4.6, 4.7, 4.8 y 4.9.

	Número empleado	Porcentaje
Virtex 5		
Slice Logic Utilization:		
Number of Slice Registers (out of 58880):	19287	32%
Number of Slice LUTs (out of 58880):	15290	25%
Number used as Logic (out of 58880):	14382	24%
Number used as Memory (out of 24320):	908	3%
Number used as SRL:	908	
Slice Logic Distribution:		
Number of LUT Flip Flop pairs used:	21321	
Number with an unused Flip Flop	2034	9%
Number with an unused LUT	6031	28%
Number of fully used LUT-FF pairs	13256	62%
Number of unique control sets:	369	
Specific Feature Utilization:		
Number of Block RAM/FIFO (out of 244):	7	2%
Number using Block RAM only:	7	
Number of BUFG/BUFGCTRLs (out of 32):	2	6%
Number of DSP48Es (out of 640):	27	4%

Tabla 4.5. Recursos utilizados de la FPGA Virtex 5, para la implementación del algoritmo VCA con todas las operaciones en punto flotante. Imagen de 36x36 píxeles y 5 endmembers a calcular.

Virtex 5	
Max. freq. (MHz)	210,438
Min. period (ns)	2,530
Latencia	9738 ciclos

Tabla 4.6. Restricciones de la implementación del algoritmo VCA con todas las operaciones en punto flotante sobre la Virtex 5. Imagen de 36x36 píxeles y 5 endmembers a calcular.

	Número empleado	Porcentaje
Virtex 5		
Slice Logic Utilization:		
Number of Slice Registers (out of 58880):	15978	27%
Number of Slice LUTs (out of 58880):	12181	20%
Number used as Logic (out of 58880):	11350	19%
Number used as Memory (out of 24320):	831	3%
Number used as SRL:	831	
Slice Logic Distribution:		
Number of LUT Flip Flop pairs used:	17419	
Number with an unused Flip Flop	1441	8%
Number with an unused LUT	5238	30%
Number of fully used LUT-FF pairs	10740	61%
Number of unique control sets:	295	
Specific Feature Utilization:		
Number of Block RAM/FIFO (out of 244):	7	2%
Number using Block RAM only:	7	
Number of BUFG/BUFGCTRLs (out of 32):	2	6%
Number of DSP48Es (out of 640):	32	5%

Tabla 4.7. Recursos utilizados de la FPGA Virtex 5, para la implementación del algoritmo VCA con la proyección de la imagen en notación entera. Imagen de 36x36 píxeles y 5 endmembers a calcular.

Virtex 5	
Max. freq. (MHz)	268,152
Min. period (ns)	2,53
Latencia	10383 ciclos

Tabla 4.8. Restricciones de la implementación del algoritmo VCA con la proyección de la imagen en notación entera sobre la Virtex 5. Imagen de 36x36 píxeles y 5 endmembers a calcular.

Los resultados muestran que el consumo de lógica es mayor en la primera implementación, esto se debe a que los bloques de operaciones matemáticas en punto flotante son más complejos y requieren más lógica que en el caso de realizar las operaciones con enteros. La frecuencia máxima de ejecución es mayor en el segundo caso lo que permite trabajar a mayor velocidad, sin embargo la latencia es mayor. En la tabla 4.9 se muestran los tiempos de ejecución de cada algoritmo sobre la FPGA.

Algoritmo	Tiempo de ejecución (ms)
Implementación 1 (fp)	0,04637
Implementación 2 (int)	0,03874

Tabla 4.9. Tiempos de ejecución de los algoritmos.

Se observa que la segunda implementación es más rápida que la primera. Se deduce de estos resultados que la implementación de la proyección de la imagen sobre el vector f en notación entera, reduce el área de ocupación en la FPGA y acelera el proceso aproximadamente un 20%.

En las tablas 4.10 y 4.11 se presentan los resúmenes de los resultados de ocupación y de tiempo de cada implementación, respectivamente

Virtex 5	**Implementación 1**	**Implementación 2**
Slice Registers	32%	27%
Slice LUTs	25%	20%
Block RAM/FIFO	2%	2%
DSP48Es	4%	5%

Tabla 4.10. Tabla comparativa de la utilización de recursos en la síntesis de las dos implementaciones sobre la Virtex 5.

	Frecuencia (MHz)	Latencia	Tiempo total (ms)
Implementación 1	210,44	9738	0,0463
Implementación 2	268,15	10383	0,0387

Tabla 4.11. Resultados de tiempo de procesado de cada una de las implementaciones del algoritmo VCA, para una imagen 36x36 píxeles de tamaño, de la que se desean extraer 5 endmembers.

5.Conclusiones y líneas futuras de investigación

En el primer capítulo de la memoria descriptiva se presentaron unos objetivos a cumplir en este trabajo fin de máster. A lo largo de los siguientes capítulos se han abordado todos estos objetivos explicando los pasos que se han seguido en la consecución de estos.

En este capítulo de conclusión, se quieren destacar los logros obtenidos, y por otra parte, mencionar algunas ideas que podrían ser interesante seguir en líneas futuras de investigación.

5.1. Conclusiones

En el primer capítulo se enfatizó en el consumo de recursos que suponen todos los algoritmos de las imágenes hiperespectrales, dada la cantidad de datos que se han de procesar. Por este motivo se destacó la importancia de la implementación de los mismos en plataformas capaces de acelerar su cómputo. Un ejemplo claro de este hecho es el trabajo que se presenta, donde se han conseguido tiempos de ejecución muy reducidos, y las posibilidades de paralelización son muy elevadas.

Del presente trabajo hay que subrayar, en primer lugar, que se ha realizado un estudio detallado del algoritmo VCA para analizar las posibles modificaciones que permitieran implementar de manera más sencilla y eficaz el algoritmo en hardware.

Se determinó una mejora crucial que fue probada a nivel de software, utilizando la herramienta Matlab, contrastándose con el algoritmo original, y que consiste en eliminar la operación de la pseudoinversa de una matriz, que es costosa y requiere de múltiples operaciones numéricas de cierta complejidad. Esta operación es sustituida por el cálculo de una base de vectores que son ortogonales y que tienen la misma función que en el caso anterior, pero cuyo cálculo es menos complejo ya que se reduce a una serie de operaciones de multiplicación, suma y división.

Introducidas estas modificaciones, se llevaron a cabo dos implementaciones a nivel RTL del algoritmo VCA, una de ellas trabajando con datos en punto flotante y la otra combinando operaciones en punto flotante y en notación entera, siendo estas últimas las mayoritarias, con el fin de tratar de disminuir los recursos lógicos que utilice la implementación.

Realizadas las implementaciones, se presentaron los resultados de síntesis de las mismas sobre una FPGA Virtex 5, en cuanto a área de ocupación sobre la FPGA y a tiempo de ejecución.

Destacar que a lo largo de la memoria se han presentado los resultados de las implementaciones del algoritmo en software y en hardware, ya que los mismos se han desarrollado en Matlab y en Verilog, para poder contrastar los resultados que se obtenían con una y otra herramienta. No se han realizado comparativas de tiempo entre uno y otra ya que no serían veraces, dadas las diferencias de recursos hardware entre las plataformas sobre las que se ejecutan los programas.

También se ha desarrollado en Matlab, un entorno de verificación con diferentes rutinas de conversión de formato de números, que junto con las instrucciones de ficheros de los archivos testbench, permiten ejecutar una correcta depuración del código.

Hasta ahora las líneas de investigación y publicaciones relacionadas con la implementación en FPGAs de algoritmos de procesamiento de imágenes hiperespectrales son escasas, por lo que este proyecto fin de carrera es pionero en este aspecto. Además los métodos que se proponen son novedosos y producen resultados contrastados a nivel numérico y muy positivos a nivel de tiempo de ejecución.

5.2. Líneas futuras de investigación

Son muchas las aplicaciones que podrían verse beneficiadas por el desarrollo de los algoritmos para imágenes hiperespectrales. Muchas de ellas requieren del procesamiento de datos en tiempo real, como pueden ser por ejemplo aplicaciones de medicina. Por este motivo el procesamiento en hardware de los algoritmos genera grandes ventajas en todas estas aplicaciones.

Como primera propuesta de continuación al trabajo realizado se plantea seguir implementando en hardware toda la cadena de algoritmos de procesamiento de imágenes hiperespectrales que se mostró en el capítulo 2. Para ello se deberá proceder de igual manera que se ha hecho en este trabajo, es decir, realizando un estudio en detalle del algoritmo, y buscando posibles modificaciones que permitan de manera más adecuada la implementación a bajo nivel.

Cualquier aplicación en tiempo real requiere de una comunicación que trabaje a frecuencias de transmisión relativamente altas y con una tasa de fallos baja, sobre todo aplicaciones precisas, donde los errores no están permitidos. En estos casos es fundamental no sólo trabajar con tecnologías de comunicación contrastadas y fiables, además de rápidas, sino que es preciso desarrollar un protocolo capaz de actuar en caso de fallos. Se propone por tanto profundizar en el estudio de las comunicaciones que permitan la transmisión de datos a altas velocidades, para que estas no sean el cuello de botella de todo el proceso.

En definitiva, la línea de desarrollo que se propone es desarrollar sistemas compactos capaces de recibir, procesar y transmitir las imágenes hiperespectrales en un

tiempo reducido, y que por tanto sean capaces de dirigirse a aplicaciones completas realizando todos los procesos necesarios.

Por supuesto, se debe continuar la investigación de algoritmos diferentes, y tratar de encontrar aquellos que ofrezcan resultados superiores en el equilibrio calidad-coste computacional.

6.Bibliografía

[1] Plaza, A., Chang, C.-I, “Impact of Initialization on Design of Endmember Extraction Algorithms,” IEEE Transactions on Geoscience and Remote Sensing, vol. 44, no. 11, pp. 3397-3407, 2006.

[2] Plaza, A., Martínez, P., Pérez, R.M., Plaza, J.. “Spatial/Spectral Endmember Extraction by Multidimensional Morphological Operations”. IEEE Transactions on Geoscience and Remote Sensing, vol. 40, no. 9, pp. 2025-2041, 2002.

[3] Heinz, D. y Chang, C.-I, "Fully constrained least squares linear mixture analysis for material quantification in hyperspectral imagery," IEEE Trans. on Geoscience and Remote Sensing, vol. 39, no. 3, pp. 529-545, March 2001.

[4] Bateson, C.A., Curtiss, B., “A method for manual endmember selection and spectral unmixing,” Remote Sensing of Environment, vol. 55, pp.229–243, 1996.

[5] Plaza, A., Valencia, D., Plaza, J. y Chang, C.-I, “Parallel Implementation of Endmember Extraction Algorithms from Hyperspectral Data”. IEEE Geoscience and Remote Sensing Letters, vol. 3, no. 3, pp. 334-338, July 2006.

[6] Plaza, A., Plaza, J., Valencia, D., “AMEEPAR: Parallel Morphological Algorithm for Hyperspectral Image Classification in Heterogeneous Networks of Workstations.” Lecture Notes in Computer Science, vol. 3391, pp. 888-891, 2006.

[7] Setoain, J., Prieto, M., Tenllado, C., Plaza, A., Tirado, F., “Parallel Morphological Endmember Extraction Using Commodity Graphics Hardware,” IEEE Geoscience and Remote Sensing Letters, vol. 43, no. 3, pp. 441-445, 2007.

[8] Pérez, R.M., Martinez, P., Plaza, A., Aguilar, P.L. “Systolic Array Methodology for a Neural Model to Solve the Mixture Problem”, in: Neural Networks and Systolic Array Design. Edited by D. Zhang and S.K. Pal. World Scientific, 2002.

[9] T. W. Anderson (2003). “An Introduction to Multivariate Statistical Analisis”. Wiley Series in Probability and Statistics, Third Edition.

[10] I. T. Jollifre (1986). “Principal Component Analysis”. Springer-Verlag, New York.

[11] L. L. Scharf (1991). “Statistical Signal Processing, Detection Estimation and Time Series Analysis”, Reading MA: Addison-Wesley.

[12] A. Green, M. Berman, P. Switzer, M. D. Craig (1994). “A transformation for ordering multispectral data in terms of image quality with implications for noise removal”, IEEE Transactions in Geoscience and Remote Sensing, vol. 26, no. 1, pp. 65-74.

[13] P. Bajorski (2004), “Simplex projection methods for selection of endmembers in hyperspectral imagery”, IEEE International Geoscience and Remote Sensing Symposium Proceedings, v. 5, pp. 3207-3210.

[14] F. Chaudhry, C. Wu, W. Liu, C. Chang, A. Plaza (2006). “Pixel purity index-based algorithms for endmember extraction from hyperspectral imagery”. Transworld Research Network, pp.30-62.

[15] M. Zortea, A. Plaza (2009). “A quantitative and comparative analysis of different implementations of N-FINDR: A fast endmember extraction algorithm”, IEEE Geoscience and Remote Sensing Letters, v. 6, nº. 4, pp. 787-791.

[16] J. Nascimento, J. Bioucas (2005). “Vertex component analysis: a fast algorithm to unmix hyperspectral data”, IEEE Transactions on Geoscience and Remote Sensing, v. 43, nº. 4, pp. 898-910.

[17] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu (2002). “An Efficient k-Means Clustering Algorithm: Analysis and Implementation”. IEEE Transactions on Pattern Analysis and Machine Learning, 24, 7, pp. 881-892.

[18] Librería Digital de Espectros USGS (United States Geological Survey), última visita: 20 de mayo 2010
http://speclab.cr.usgs.gov/spectral.lib06/ds231/datatable.html

[19] Librería espectral ASTER, propiedad de la NASA, última visita: 6 de junio de 2010
http://speclib.jpl.nasa.gov/

[20] S. Robila, A. Gershman (2005). "Spectral Matching Accuracy in Processing Hyperspectral Data". Department of Computer Science. Montclair State University.

[21] M. Canty (2006). "Image Analysis, Classification and Change Detection in Remote Sensing", Taylor & Francis.

[22] Lavenier, D., Fabiani, E., Derrien, S., Wagner, C., "Systolic array for computing the píxel purity index (PPI) algorithm on hyper spectral images".

[23] C. Chang, C. Wu, C. Tsai (2011). "Random N-finder (N-FINDR) endmember extraction algorithms for hyperspectral imagery", IEEE Transactions of Image Processing, v. 20, nº. 3, pp. 641-655.

[24] ANSI/IEEE Std 754-1985, IEEE Standard for Binary Floating-Point Arithmetic, 1985.

[25] D. Goldberg (1991). "What every computer scientist should know about floating point arithmetics". ACM Computing Surveys (CSUR), volume 23 issue 1.

[26] IPCores de Xilinx, última visita: 6 de octubre de 2010
http://www.xilinx.com/ipcenter/