Arquitectura hardware para la detección de anomalías en imágenes hiperespectrales

Julián Caba[1], María Díaz[2], Jesús Barba[1], Raúl Guerra[2], Soledad Escolar[1], Sebastián López[2], Fernando Rincón[1] y Juan Carlos López[1]

Resumen— **El procesamiento de datos a bordo en vehículos aéreos no tripulados (UAV) con sensores de imágenes ha cobrado un reciente impulso en el campo de la sensorización remota. En este contexto, la detección de anomalías hiperspectrales ha recibido una atención especial, cuyo principal objetivo es identificar de forma no supervisada eventos anormales. Sin embargo, el procesamiento de dichas imágenes plantea varios retos. En este trabajo se ha desarrollado una arquitectura hardware que asegura el procesamiento en aplicaciones sensibles al tiempo y limitadas por la escasez de recursos hardware. En este sentido, se ha implementado el detector HW-LbL-FAD en hardware reconfigurable para obtener un rendimiento en tiempo real. En concreto, se ha seleccionado una FPGA de coste optimizado (ZC7Z020-CLG484) para implementar nuestra solución cuyos resultados dibujan un buen compromiso entre las tres características siguientes: rendimiento temporal, consumo energético y coste. Los resultados indican que la arquitectura desarrollada es capaz de procesar imágenes hiperespectrales de 825x1024 píxeles y 160 bandas en 0,51 segundos con un presupuesto de energía de 1,3 vatios y un coste del dispositivo de unos 150 euros.**

Palabras clave— **FPGAs, hyperspectral imaging, anomaly detection, line-by-line performance, real-time, High-Level Synthesis, low-power.**

I. Introducción

En los últimos años, la detección de anomalías ha sido ampliamente estudiada en el área de análisis de datos hiperspectrales, donde su popularidad es cada vez mayor en diferentes aplicaciones debido a la capacidad de detectar eventos anómalos u objetivos artificiales de forma no supervisada [1]. Por norma general, las anomalías se consideran píxeles no abundantes en una escena cuyo espectro es significativamente distinto al de su entorno o del patrón de fondo predominante. Por lo tanto, uno de los fundamentos del problema de la detección de anomalías es la identificación de objetivos deseados que se desconocen de antemano y cuya existencia podría ser indicativa de un comportamiento sospechoso. Esta falta de conocimiento previo convierte la detección de espectros anómalos en una cuestión esencial en aplicaciones militares y civiles, como la defensa y la vigilancia, la monitorización medioambiental, la detección de desastres naturales raros o los estudios agrícolas entre otros [2], [3].

La detección de anomalías es una tarea muy difícil, cuyo éxito depende de la representación precisa de un fondo desconocido y heterogéneo. Para ello, se han propuesto en la literatura varias soluciones algorítmicas. Las más estudiadas se basan en enfoques estadísticos bajo el supuesto de una distribución multivariante gaussiana. En este sentido, el método Reed-Xiaoli (RX) [4] se considera el referente en el campo de la detección de anomalías y en la literatura se pueden encontrar diversas variantes mejoradas. No obstante, la aparición de modelos de detección comprimida y de representación colaborativa ha permitido desarrollar con éxito otros métodos no basados en RX que no asumen una función de probabilidad normal en patrones de fondo muy heterogéneos [5], [6], [7], [8]. Otras propuestas se basan en la reducción de dimensiones y la extracción de características para eliminar las correlaciones entre bandas espectrales [9]. Además, los recientes avances en *deep learning* y redes neuronales han irrumpido en el área [10], [11].

La mayoría de las soluciones algorítmicas mencionadas alcanzan buenos niveles de precisión en cuanto a la detección se refiere, pero a cambio de aumentar la complejidad de cálculo de las operaciones implicadas: uso intensivo de memoria, escalabilidad limitada y/o bajo grado de paralelismo [12]. Este hecho impide que los UAVs incorporen estas soluciones en el procesamiento de imágenes a bordo. Además, estas plataformas aéreas están limitadas en términos de capacidades computacionales, presupuesto de energía y almacenamiento de datos. Por lo tanto, el procesamiento en tierra ha sido la solución principal para el manejo de datos hiperespectrales. Sin embargo, las tasas de adquisición de datos son cada vez mayores, mientras que la transmisión de enormes volúmenes de datos es cada vez más inviable para la realización de aplicaciones de baja latencia que conlleven toma de decisiones [13]. Por ello, el procesamiento a bordo se ha convertido en una potencial solución para estos escenarios [14].

En este trabajo se ha desarrollado una arquitectura hardware para dispositivos basados en lógica reconfigurable del algoritmo HW-LbL-FAD. El algoritmo HW-LbL-FAD es un detector de anomalías basado en proyecciones ortogonales y ha sido diseñado para cumplir con las restricciones impuestas por las aplicaciones de teledetección actuales basadas en escáneres *pushbroom/whiskbroom*. A continuación se listan las contribuciones de este trabajo.

- Diseño e implementación del algoritmo *HW-LbL-FAD* en un dispositivo de lógica reconfigurable mediante uso de síntesis de alto nivel (High-Level Synthesis, HLS).

[1]Dpto. Tecnologías y Sistemas de Información, Universidad de Castilla-La Mancha, 13071 Ciudad Real, Spain. e-mail: {julian.caba, jesus.barba, soledad.escolar, fernando.rincon, juancarlos.lopez}@uclm.es

[2]Instituto Universitario de Microelectrónica Aplicada (IUMA), 35017 Las Palmas de Gran Canaria, Spain. e-mail: {mdmartin, rguerra, seblopez}@iuma.ulpgc.es

- Análisis de la implementación desarrollada de acuerdo a la cantidad de datos que pueden ser procesados en paralelo.
- Comparativa de la solución propuesta en este trabajo frente a otras soluciones basadas en FPGA del estado del arte.

El resto de este documento está organizado de la siguiente forma. La sección II incluye un breve resumen sobre del algoritmo HW-LbL-FAD. La sección III incluye una explicación exhaustiva sobre la arquitectura hardware desarrollada para la ejecución del algoritmo HW-LbL-FAD sobre lógica reconfigurable. La sección IV evalúa el detector de anomalías propuesto mediante el análisis de la arquitectura hardware implementada en un dispositivo ZC7Z020. La sección V recoge una discusión sobre el rendimiento y la utilización de recursos hardware de la arquitectura hardware propuesta en comparación con otros detectores de anomalías del estado del arte también implementados en hardware reconfigurable. Por último, la sección VI recoge las principales conclusiones de este trabajo.

II. Descripción del algoritmo HW-LbL-FAD

El algoritmo HW-LbL-FAD es un detector de anomalías basado en el subespacio que se basa en el concepto del modelo de mezcla lineal (LMM) para tratar el problema de la detección de anomalías. Partiendo de la premisa de que los espectros anómalos y los del fondo se encuentran en subespacios dimensionales diferentes, el LMM puede reescribirse como:

$$\mathbf{r}_\mathrm{j} = \sum_{n=1}^{p} \mathbf{b}_\mathrm{n} \cdot a_{\mathrm{j,n}} + \mathbf{s} \cdot a_{\mathbf{s}\mathrm{j}} + \mathbf{n}_\mathrm{j} \qquad (1)$$

donde $\mathbf{b}_\mathrm{n}$ representa el fondo n, $\mathbf{s}$ es la firma de destino deseada, $a_\mathbf{s}$ es el factor de abundancia de $\mathbf{s}$ en el píxel $\mathbf{r}_\mathrm{j}$ y $\mathbf{n}_\mathrm{j}$ representa el ruido contenido en el píxel de la imagen $\mathbf{r}_j$.

Cabe destacar que los espectros anómalos no se conocen de antemano, pero una característica distintiva es que no coinciden con el patrón de fondo y, por tanto, no pueden ser representados con precisión por las muestras de fondo. Por lo tanto, la detección de anomalías basada en el subespacio se basa en la idea de que las entidades anómalas se detectan mejor en el subespacio ortogonal que abarca la distribución del fondo. En este contexto, el índice de separación de la proyección para una imagen $\mathbf{r}_\mathrm{j}$ se calcula como:

$$\mathbf{d} = (\mathbf{r}_\mathrm{j} - \hat{\boldsymbol{\mu}}_b)' \cdot \mathbf{P} \cdot (\mathbf{r}_\mathrm{j} - \hat{\boldsymbol{\mu}}_b) \qquad (2)$$

donde $\hat{\boldsymbol{\mu}}_\mathbf{b}$ es el píxel medio estimado de las muestras de fondo y $\mathbf{P}$ es la matriz que proyecta los datos en el subespacio ortogonal a uno abarcado por las muestras de fondo.

Sobre esta base, la cuestión de la detección de espectros anómalos puede dividirse en dos tareas principales, uno es el modelado del patrón de fondo y el otro es el cálculo del subespacio ortogonal abarcado por éste para marcar aquellos píxeles con la mayor proyección como posibles anomalías. Por tanto, la funcionalidad del algoritmo HW-LbL-FAD se describe en cuatro etapas de cálculo para abordar las dos tareas mencionadas. La figura 1 muestra el diagrama de flujo de datos de las etapas de cálculo de HW-LbL-FAD, en el que también se representan las entradas y salidas. Las dos primeras etapas se realizan de forma secuencial, mientras que las dos últimas se realizan en paralelo. Estas etapas de cálculo se describen en los algoritmos 1 y 2.

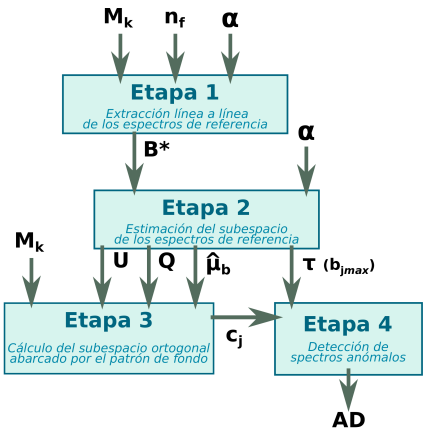

Fig. 1: Etapas del algoritmo HW-LbL-FAD.

Algoritmo 1 Conjunto de operationes

Inputs: $\mathbf{M}_k = [\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{BS}], \alpha$
Outputs: $\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_p]$ {Characteristic pixels}; $\hat{\boldsymbol{\mu}}$ {Average Pixel}; $\mathbf{Q} = [\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_p]$ {Orthogonal vectors}; $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_p]$ {Orthonormal vectors}; τ {Threshold}
Algorithm:
1: Average pixel: $\hat{\boldsymbol{\mu}}$;
2: Centralization: $\mathbf{C} = \mathbf{M}_k - \hat{\boldsymbol{\mu}}$; exit = 0 $j = 1$ **to** BS
3: Brightness: $\mathbf{b}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$;
4: Maximum Brightness: $j_{max} = \mathrm{argmax}(\mathbf{b}_j)$; $\frac{b_{j\max}}{(\mathbf{r}_{j\max} - \hat{\boldsymbol{\mu}})' \cdot (\mathbf{r}_{j\max} - \hat{\boldsymbol{\mu}})} \cdot 100 < \alpha$
5: Stop Condition: exit = 1
6: Extracted pixels: $\mathbf{e}_n = \mathbf{r}_{j_{max}}$;
7: $\mathbf{q}_n = \mathbf{c}_{j_{max}}$;
8: $\mathbf{u}_n = \mathbf{q}_n / b_{j_{max}}$;
9: Projection: $\mathbf{v}_n = \mathbf{u}'_n \cdot \mathbf{C}$;
10: Subtraction: $\mathbf{C} = \mathbf{C} - \mathbf{q}_n \cdot \mathbf{v}_n$;
11: $\tau = b_{j\max}$

Algoritmo 2 Algoritmo HW-LbL-FAD.

Inputs: $\mathbf{HI} = [\mathbf{M}_1, \mathbf{M}_2, ..., \mathbf{M}_{\frac{nr \cdot nc}{BS}}], n_f, \alpha$
Outputs: $\mathbf{AD} = [\mathbf{x}_{11}, \mathbf{x}_{12}, ..., \mathbf{x}_{kj}]$
Algorithm:
Stage 1: k = 1 **to** n_f
1: $\mathbf{E}_\mathrm{k} = \mathbf{Algorithm\ 1}(\mathbf{M}_\mathrm{k}, \alpha)$
2: $\mathbf{B^*} = [\mathbf{B^*}, \mathbf{E}_\mathrm{k}]$;
Stage 2:
3: $[\hat{\boldsymbol{\mu}}_b, \mathbf{Q}, \mathbf{U}, \tau] = \mathbf{Algorithm\ 1}(\mathbf{B^*}, \alpha)$
Stage 3: Applied to each new received frame, $\mathbf{M}_k$, $k > n_f$ j = 1 **to** BS
4: Centralization: $\mathbf{c}_j = \mathbf{r}_j - \hat{\mu}_b$ n = 1 **to** p
5: Projection: $v = \mathbf{U}'_n \cdot \mathbf{c}_j$
6: Subtraction: $\mathbf{c}_j = \mathbf{c}_j - \mathbf{Q}_n \cdot v$
Stage 4:
7: Brightness AD: $\mathbf{d}_j = \mathbf{c}'_j \cdot \mathbf{c}_j$ $\mathbf{d}_j \leq 1{,}5 \cdot \tau$ $(\mathbf{x}_{kj} = 0)$ $(\mathbf{x}_{kj} = 1)$

En este sentido, el algoritmo HW-LbL-FAD muestra algunas características únicas que lo convierten en un buen candidato:

- *Baja complejidad computacional y alto grado de paralelismo de las operaciones involucradas.* Las

diferentes etapas de cálculo definidas en el algoritmo HW-LbL-FAD utilizan un conjunto de operaciones básicas, como son el cálculo de la matriz inversa o la extracción de valores y vectores propios. Esto facilita su posterior implementación en hardware, reuciendo la cantidad de recursos hardware necesarios.

- *Reducción de los recursos de computación.* El algoritmo HW-LbL-FAD es un detector de anomalías basado en dos procesos diferenciados: (1) el modelado del fondo y; (2) la detección de potenciales espectros anómalos. Ambos procesos se basan en el mismo método matemático y, de hecho, comparten el conjunto de operaciones básicas. En consecuencia, los recursos dedicados a la implementación de los operadores de hardware pueden ser compartidos por estos dos procesos, ya que el segundo proceso depende del primero, es decir, no son paralelos entre sí.
- *Rendimiento línea a línea.* El algoritmo HW-LbL-FAD es capaz de procesar bloques de píxeles de imágenes sin tener en cuenta ningún requisito de alineación espacial. Además, el algoritmo analiza los bloques como si fueran independientes entre sí. Por lo tanto, el método seguido por el algoritmo HW-LbL-FAD permite reducir la cantidad de datos que se deben almacenar, minimizando así los recursos de memoria necesarios y acelerando el proceso de análisis datos.
- *Notación de punto fijo.* El conjunto de operaciones puede implementarse sin problemas utilizando tanto la notación de punto flotante como la de punto fijo, sin incurrir en una pérdida significativa de precisión en los resultados de detección, como se analizó en [15][16]. Por lo tanto, el algoritmo HW-LbL-FAD puede implementarse adecuadamente en plataformas de computación, como FPGAs y GPUs, y seguir aprovechando las características específicas de la arquitectura subyacente, manteniendo excelentes niveles de rendimiento.

III. Arquitectura hardware para el algoritmo HW-LbL-FAD

El algoritmo HW-LbL-FAD se ha implementado en una plataforma basada en FPGA utilizando el *toolchain* de Xilinx. La tecnología HLS permite a los ingenieros reducir el tiempo de desarrollo y la complejidad de los diseños de hardware gracias al uso de lenguajes de alto nivel (HLL), como C o C++, para describir la funcionalidad, posteriormente se traduce en un modelo RTL (Register Transfer Level). En este trabajo, se ha hecho uso de este tipo de herramientas para aumentar la productividad y acortar los ciclos de desarrollo. Para ello, los modelos HLS de algunas operaciones utilizadas por el algoritmo HW-Lbl-FAD se han heredado de un desarrollo previo, descrito en [15], donde el objetivo de esas operaciones es acelerar la compresión de imágenes hiperespectrales. Para estos componentes heredados, el principal trabajo a realizar fue su adaptación para que se ajustaran a los requisitos funcionales y no funcionales de la implementación del algoritmo HW-Lbl-FAD. Este proceso de adaptación de componentes es más corto y menos laborioso que abordar el diseño de uno nuevo gracias a los beneficios que aportan las herramientas HLS. Sin embargo, el proceso no está completamente exento de ciertos ajustes que permitan reducir la latencia y aumentar el rendimiento.

Por lo tanto, la arquitectura diseñada para el algoritmo HW-Lbl-FAD es una combinación de módulos heredados y rediseñados junto con otros nuevos, que implementan el procesamiento de datos hiperespectrales. Una vez modelados y probados dichos módulos, fue necesario orquestar su funcionamiento para implementar el comportamiento de las diferentes etapas del detector de anomalías. El enfoque seguido para dar solución a estos dos retos fue implementar una máquina de estados finita (FSM, Finite State Machine) en VHDL que activa los selectores correspondientes de acuerdo a la etapa en ejecución (trapecios azules en las figuras 2, 3 y 4), así como cada acelerador hardware especializado en el momento adecuado.

Desde el punto de vista del rendimiento, la aritmética se ha adaptado a la precisión de punto fijo con el fin de optimizar la implementación sobre la tecnología FPGA. En trabajos anteriores [15], [16] se demostró que la simplificación de las operaciones aritméticas empleando aritmética de enteros y desplazamiento de bits [17] no afectaba significativamente a la precisión de los resultados, incluso aumenta la calidad de la salida en algunos escenarios. Así, aplicando lo aprendido en trabajos anteriores, se ha fijado el tamaño de la información hiperespectral en un máximo de 12 bits, que es el tamaño de bit más común en las aplicaciones de teledetección [18], [19] y, como se muestra en [15], también proporciona buenos resultados de precisión.

En aras de la claridad, las siguientes secciones describen la implementación hardware de las cuatro etapas que componen el algoritmo HW-LbL-FAD. Cabe recordar que la arquitectura es una sola, pero se ha dividido en tres figuras, cada una de las cuales resalta los módulos y conexiones que intervienen en cada una de las etapas del algoritmo HW-LbL-FAD.

A. Etapa 1: Extracción línea por línea de los espectros de referencia

La primera etapa del algoritmo HW-LbL-FAD consiste en la selección de un conjunto de espectros de referencia representativos del fondo. Para ello, las primeras n_f líneas hiperespectrales; $\mathbf{M}_k$, $k = [1, ..., n_f]$; capturadas por el sensor hiperespectral se procesan de forma independiente para seleccionar los espectros más diferentes dentro de ellas, $\mathbf{B^*} = [\mathbf{E}_1, ..., \mathbf{E}_{\mathrm{n_f}}]$ (ver líneas 1-4 del Algoritmo 2).

La figura 2 resalta las operaciones realizadas en la primera etapa del algoritmo HW-LbL-FAD para extraer los píxeles de referencia del fondo. En primer lugar, se almacena el bloque hiperespectral original, $\mathbf{M}_k$, en una memoria interna (*SBuffer*) y se obtiene

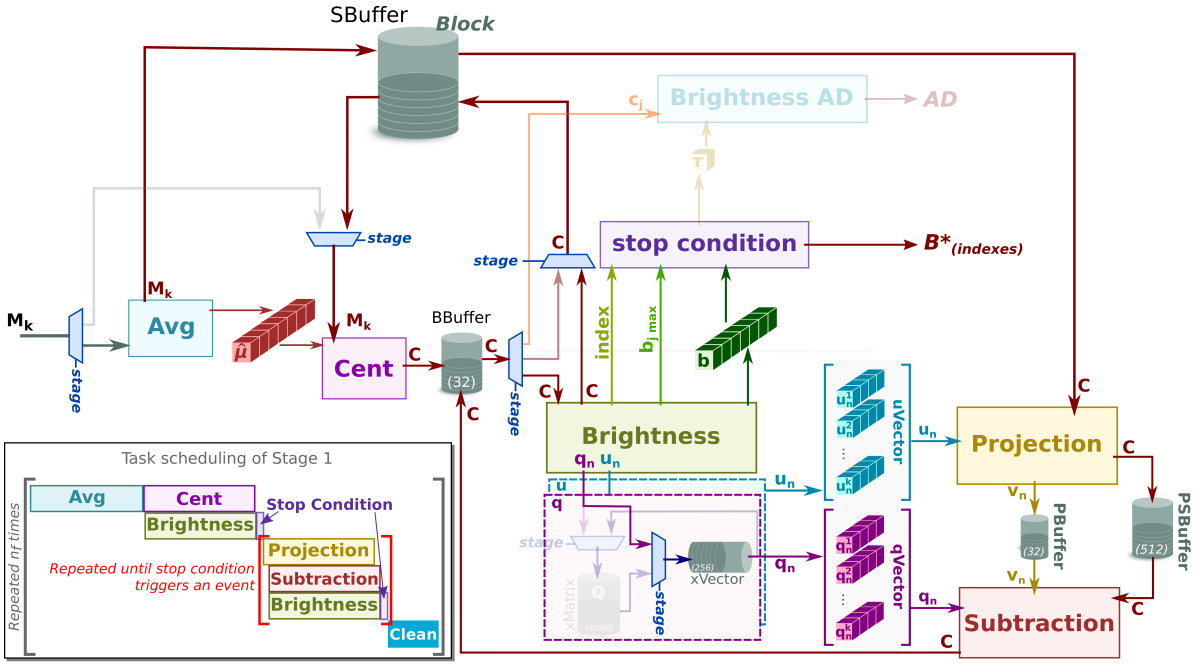

Fig. 2: Etapa 1: Extracción línea por línea de los espectros de referencia.

su centroide o píxel promedio ($\hat{\boldsymbol{\mu}}$). Ambas operaciones son realizadas por el módulo *Avg*, que es capaz de computar varias bandas de un bloque hiperespectral en paralelo (línea 1 del Algoritmo 1). Una vez que el módulo *Avg* termina, el módulo *Cent* puede iniciar la centralización del bloque hiperespectral original leyéndolo de *SBuffer*, es decir, esta operación consiste en restar a cada píxel hiperespectral del bloque original, $\mathbf{M}_k$, el píxel medio que ha sido calculado por el módulo *Avg* (línea 2 del Algoritmo 1). Por lo tanto, el componente *SBuffer* debe garantizar una profundidad suficiente para almacenar un bloque hiperespectral. El resultado del módulo *Cent* se escribe en el componente *BBuffer*, cuya profundidad es pequeña en comparación con la del *SBuffer*. Esta profundidad se debe a que, tan pronto como la salida del módulo *Cent* esté lista, el siguiente módulo puede consumirla. Por lo tanto, el bucle del Algoritmo 1 puede comenzar sin esperar a que se complete el paso de centralización de bloques.

A diferencia de los módulos *Avg* y *Cent*, el modelo HLS del componente *Brightness* tuvo que ser rediseñado para implementar la condición de parada dinámica requerida por el algoritmo HW-Lbl-FAD. Esta característica no era soportada por la versión desarrollada en [15], ya que el número de iteraciones se fijaba en el momento de diseño. El número de veces que se ejecutan los pasos *Brightness*, *Projection* y *Subtraction* depende del píxel hiperespectral más brillante seleccionado durante la iteración actual, su brillo obtenido en la primera iteración y el parámetro de entrada α (línea 8 del Algoritmo 1). Todos estos valores, excepto el parámetro α que se fija en tiempo de diseño, son proporcionados por el módulo *Brightness*. Por lo tanto, el módulo *Brightness* almacena todos los valores de brillo calculados en la primera iteración en una BRAM (elemento **b** en verde de la Figura 2) y proporciona al módulo *Stop condition* el valor del píxel hiperespectral más brillante (b_{jmax}) y su índice dentro del bloque hiperespectral (*index*), que se está procesando actualmente. El módulo *stop condition* determina el número de iteraciones a realizar, es decir, implementa el control de la variable *exit* del Algoritmo 1 y la sentencia *if*. En este sentido, la condición *if* implica que al menos el módulo *brightness* se ejecute dos veces, ya que en la primera iteración la condición se evalúa a falso porque se está comparando el mismo brillo y no hay diferencia. El módulo *stop condition* también se encarga de proporcionar los índices de los píxeles hiperespectrales que corresponden a los espectros de referencia del fondo ($\boldsymbol{B}^*$), por lo que devuelve un índice por iteración.

Siempre que el módulo *Stop condition* no provoque la salida del bucle, el resto de las operaciones dentro del bucle del Algoritmo 1 pueden llevarse a cabo. Así, los vectores de proyección ortogonales $\boldsymbol{q_n}$ y $\boldsymbol{u_n}$ se obtienen, en consecuencia, utilizando el píxel más brillante calculado por el módulo *Brightness* (líneas 12 y 13 del Algoritmo 1) y se almacenan en FIFOs separadas (representados como *xVector* en la Figura 2): *qVector* y *uVector*, respectivamente.

B. *Etapa 2: Estimación del subespacio de los espectros de referencia*

Dado que los bloques hiperespectrales se procesan de forma independiente en la *Etapa 1* descartando cualquier restricción de alineación espacial, $\mathbf{B^*}$ comprende varios espectros iguales. Como consecuencia, se requiere obtener un subconjunto de los píxeles más diferentes dentro de $\mathbf{B^*}$ que definan mejor la distribución del fondo, es decir, modelar el fondo (ver línea 5 del algoritmo 2). Para ello, se aplica de nuevo el conjunto de operaciones básicas, aunque la matriz de entrada, $\mathbf{M}_k$, se sustituye ahora por $\mathbf{B^*}$ cuyas columnas recogen los vectores de referencia de fondo extraídos de cada uno de los primeros n_f bloques.

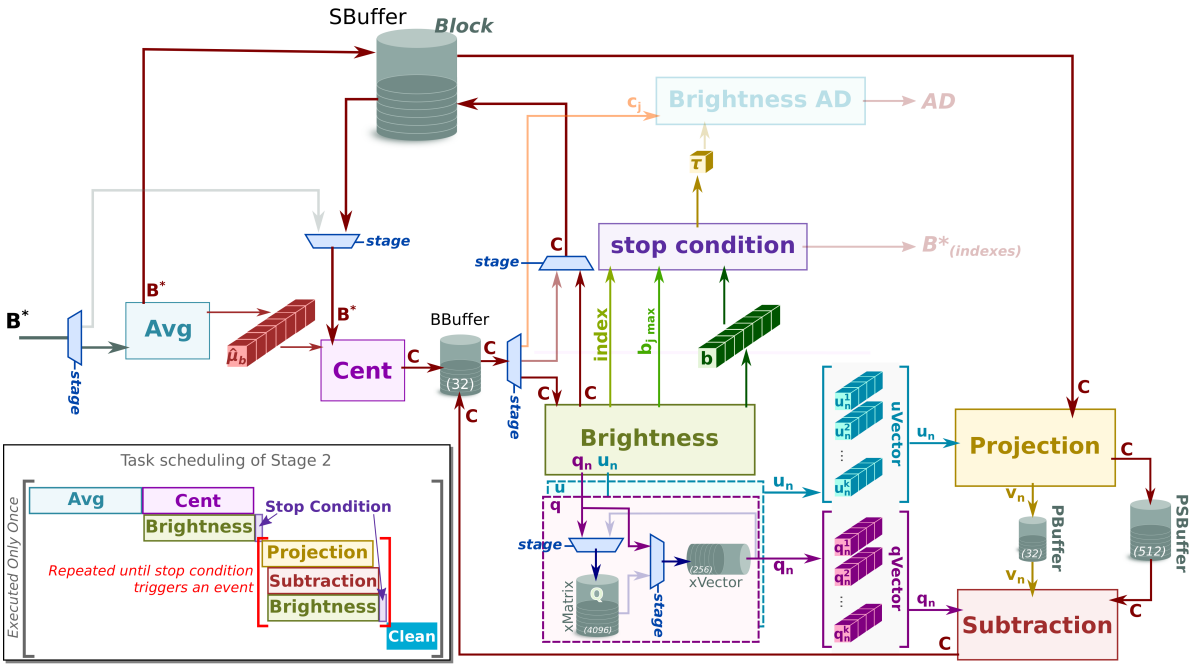

Fig. 3: Etapa 2: Estimación del subespacio de los espectros de referencia.

Como salida, se obtiene el subespacio del fondo compuesto por vectores ortogonales, **Q** y **U**, cuya definición evita el cálculo implícito de la matriz de proyección ortogonal, **P**, en la *Etapa 3* del algoritmo, que es una tarea muy exigente computacionalmente.

En cuanto a la arquitectura hardware, aunque parte de los módulos utilizados en la arquitectura hardware del algoritmo HW-LbL-FAD fueron también utilizados en el caso de uso de compresión de imágenes hiperespectrales presentado en [15]. El algoritmo HW-LbL-FAD implica nuevos pasos, no contemplados en la compresión de imágenes hiperespectrales, que deben ser realizados por el acelerador. Por ello, desde un principio el modelado del fondo y la detección de posibles espectros anómalos se diseñaron para utilizar el mismo conjunto de operaciones básicas que, a la hora de la implementación física, se traduce en recursos informáticos compartidos. Además, se ha tomando como base la ruta de datos de la *Etapa 1*, donde para soportar la nueva funcionalidad se debe dotar a la arquitectura de un enrutamiento dinámico, cuyas variaciones se realizan mediante pequeños selectores (gobernados por las señales de control de la FSM) cuyo canal de entrada o salida depende de la etapa actual. De este modo, los mismos operadores hiperespectrales pueden ser reutilizados a través de las diferentes etapas.

En esta segunda etapa, el bloque hiperespectral de entrada, $\boldsymbol{B^*}$, es el espectro de referencia con los píxeles extraídos de la *Etapa 1*. A diferencia de la etapa anterior, la segunda sólo opera con un único bloque hiperespectral en lugar de $\boldsymbol{n_f}$ bloques. Sin embargo, es muy probable que el tamaño del bloque sea diferente en ambas etapas en cuanto a tamaño se refiere. Este hecho implica extender el bloque hiperespectral procesado por la segunda etapa hasta los 1024 píxeles, repitiendo en orden sus datos, con el fin de evitar implementar una solución en la que el tamaño del bloque sea variable, ya que provoca un aumento de los recursos de hardware y rompe las posibles optimizaciones hardware, es decir, la arquitectura propuesta mantiene el tamaño del bloque en todas las etapas a 1024 píxeles hiperespectrales.

La figura 3 muestra una visión general de la solución basada en FPGA para la segunda etapa. Cabe destacar que el *datapath* se ha modificado ligeramente introduciendo dos variaciones principales en la arquitectura. En primer lugar, los vectores de proyección ortogonal $\boldsymbol{u_n}$ y $\boldsymbol{q_n}$ extraídos en esta etapa, $\boldsymbol{B^*}$, se almacenan en memorias internas, *uMatrix* y *qMatrix*, y son consumidos también por los módulos *Projection* y *Subtraction*, respectivamente. En otras palabras, la salida del módulo *Brightness* alimentan a los siguientes módulos como si se tratara de la primera etapa y, además, se almacena una copia de estas salidas para ser utilizadas por las siguientes etapas. También se almacena el número de vectores de proyección ortogonal, $\boldsymbol{u_n}$ y $\boldsymbol{q_n}$, almacenados. En segundo lugar, el otro cambio se refiere al módulo *Stop condition* que no proporciona ninguna salida en este caso, pero si almacena en un registro el brillo máximo restante, τ, del bloque hiperespectral que representa al fondo, $\boldsymbol{B^*}$.

De forma similar a la *Etapa 1*, cuando se procesa el bloque hiperespectral, $\boldsymbol{B^*}$, se limpian algunas memorias internas. Por otro lado, hay memorias internas que almacenan datos para las siguientes etapas; las FIFOs *qMatrix* y *uMatrix*, representados como *xMatrix* en la Figura 3, almacenan los vectores $\boldsymbol{Q}$ y $\boldsymbol{U}$, respectivamente, el centroide del bloque hiperespectral que representa al fondo, $\hat{\boldsymbol{\mu}}_{\boldsymbol{b}}$, y el brillo máximo restante, τ, utilizado como umbral en la última etapa. El cuadro inferior de la Figura 3 muestra cómo las etapas son ejecutadas a lo largo del tiempo no se ven alteradas respecto a la *Etapa 1*.

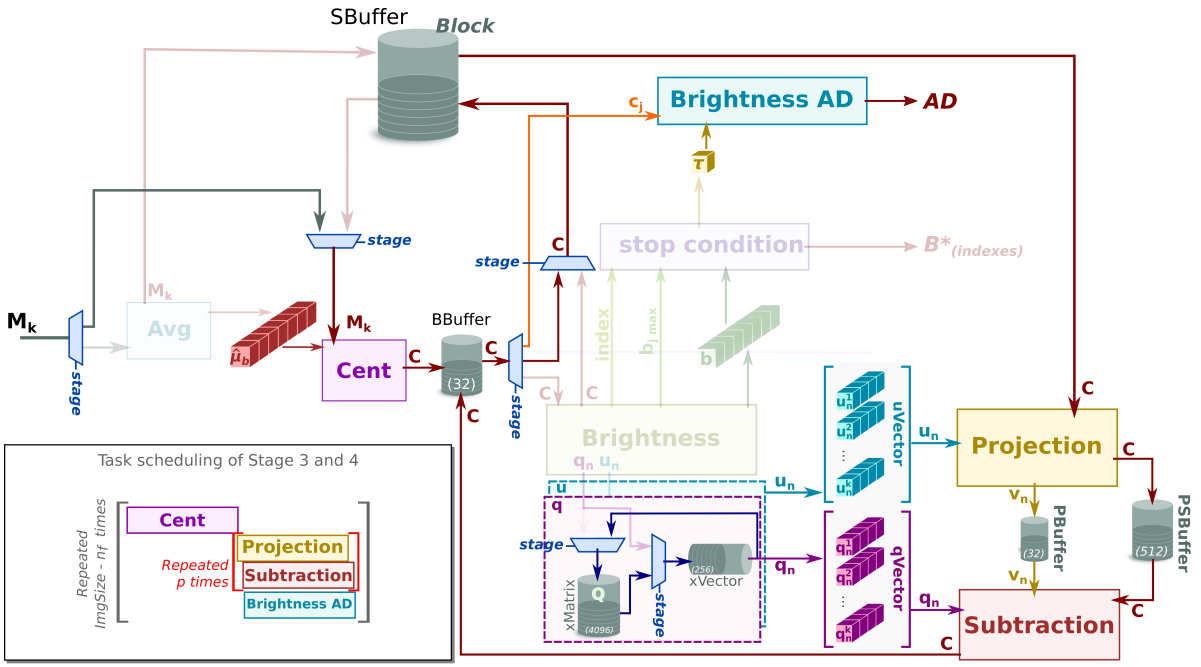

Fig. 4: Etapa 3 y 4: Cálculo del subespacio ortogonal abarcado por el patrón de fondo y detección de espectros anómalos.

C. *Etapa 3 y 4: Cálculo del subespacio ortogonal abarcado por el patrón de fondo y detección de espectros anómalos*

La *Etapa 3* del algoritmo HW-LbL-FAD se centra en el cálculo de la matriz del subespacio ortogonal, $\mathbf{P}$. Sin embargo, este cálculo es costoso desde el punto de vista computacional, ya que implica un cálculo de la matriz inversa cuya dimensión depende directamente del número de muestras de fondo, p. Sobre esta base, el algoritmo HW-LbL-FAD tiene en cuenta este problema y ofrece una alternativa optimizada. En este sentido, el índice de separación de la proyección, d, (ver ecuación 2), coincide en realidad con el brillo de cada píxel de la imagen, $\mathbf{r}_j$, después de ser sustraída toda la información perteneciente al fondo y por lo tanto, es ortogonal al subespacio abarcado por la distribución del fondo. Por lo tanto, también es equivalente a aplicar el método de ortogonalización de Gram-Schmidt para obtener la componente ortogonal de cada píxel de la imagen, $\mathbf{r}_j$, al subespacio abarcado por los vectores $\mathbf{Q}$ y $\mathbf{U}$ obtenidos en *Etapa 2*, que sí son una representación ortogonal de la distribución del fondo (ver líneas 6-11 del algoritmo 2). Como se puede observar, seguir esta estrategia permite reutilizar el conjunto de operaciones básicas ya definidas, lo que conduce a una optimización de los recursos hardware y a un menor *time-to-market*.

Desde el punto de vista de la arquitectura, esta etapa es la que más difiere del resto, ya que sólo utiliza tres operadores hiperespectrales (ver Figura 4). En primer lugar, el nuevo bloque hiperespectral, $\boldsymbol{M_k}$, se centraliza utilizando el centroide obtenido en la *Etapa 2*, $\hat{\boldsymbol{\mu}}_{\boldsymbol{b}}$. Así, el centroide ya no se calcula cuando se procesa un nuevo bloque hiperespectral en busca de spectros anómalos. Por lo tanto, el módulo *Avg* se omite y el módulo *Cent* es en realidad la primera operación que se realiza (línea 7 del Algoritmo 2).

Una vez centralizado el bloque hiperespectral, se almacena en *SBuffer* para ser procesado posteriormente por los módulos *Projection* y *Substraction*, que forman el nuevo cuerpo del bucle de esta etapa (líneas 9 y 10 del Algoritmo 2). Ahora, el número de iteraciones no es dinámico, sino que se conoce de antemano y coincide con el número de vectores ortogonales extraídos en la *Etapa 2*, p. Por lo tanto, ni los módulos *Brightness* ni *Stop condition* están habilitados en esta etapa y, por lo tanto, el módulo *Brightness* no genera ningún vector ortogonal nuevo, $\boldsymbol{u_n}$ y $\boldsymbol{q_n}$. En cualquier caso, los módulos *Projection* y *Subtraction* obtienen la imagen proyectada, $\mathbf{v}_n$, y la imagen sustraída, $\mathbf{C}$, respectivamente, del bloque hiperespectral centralizado, utilizando los vectores ortogonales que fueron calculados en la *Etapa 2* y almacenados en *qMatrix* y *uMatrix*. Cabe mencionar que estos vectores ortogonales deben ser recordados para los siguientes bloques, por lo que cuando los módulos *Projection* y *Subtraction* leen un dato a través de los buffers *uVector* y *qVector*, respectivamente, dicho dato se vuelve almacenar en las correspondientes memorias internas *uMatrix* y *qMatrix* (ver flechas azul oscuro de la Figura 4).

A diferencia de la *Stapa 1* y *2*, la imagen sustraída obtenida en la última iteración del bucle no se descarta. Por lo tanto, la salida generada por el módulo *Subtraction* en la última iteración alimenta a *etapa 4* (ver flecha naranja de la Figura 4). Esta etapa está compuesta por un único módulo llamado *Brightness AD*, que es el encargado de obtener el mapa de anomalías mediante la identificación de los píxeles anómalos. El módulo *Brightness AD* procesa el bloque $\boldsymbol{c_j}$ para obtener la luminosidad de cada uno de los píxeles hiperespectrales que componen la última imagen sustraída y los compara con el umbral, $\boldsymbol{\tau}$, obtenido en la *Etapa 2* (líneas 12 y 13 del Algoritmo 2, respectivamente). La salida de este módulo es

Tabla I: Recursos utilizados por el acelerador HW-LbL-FAD algorithm en un dispositivo Xilinx ZynQ-7020 y FPS alcanzado.

PEs	BRAM18K	DSP48E	FlipFlops	LUTs	FPS	
					(100 MHz)	*(143 MHz)*
1	214 *(76,43 %)*	14 *(6,36 %)*	8.073 *(7,59 %)*	6.744 *(12,68 %)*	84	130
2	183 *(65,36 %)*	22 *(10 %)*	8.624 *(8,11 %)*	7.470 *(14,08 %)*	169	259
4	191 *(68,21 %)*	38 *(17,27 %)*	9.981 *(9,38 %)*	9.115 *(17,13 %)*	338	518
5	193 *(68,93 %)*	46 *(20,9 %)*	10.656 *(10,01 %)*	9.940 *(18,68 %)*	424	606
8	197 *(70,36 %)*	70 *(31,82 %)*	12.666 *(11,9 %)*	12.411 *(23,33 %)*	592	900
10	205 *(73,21 %)*	86 *(39,09 %)*	14.787 *(13,9 %)*	14.856 *(27,92 %)*	697	1.055
16	193 *(68,93 %)*	134 *(60,91 %)*	18.433 *(17,32 %)*	18.724 *(35,2 %)*	948	1.423
20	197 *(70,36 %)*	166 *(75,45 %)*	23.071 *(21,68 %)*	23.493 *(44,16 %)*	1077	1.609

un mapa de anomalías, donde cada bit informa del estado de un píxel; 1 indica que es un píxel anómalo (el valor de brillo es mayor que el umbral) y 0 indica que el píxel no tiene espectros anómalos.

La *etapa 3* y *4* pueden realizar sus tareas en paralelo una vez que el bloque hiperespectral está centralizado. El recuadro inferior de la Figura 4 muestra la programación de estas tareas, los módulos *Projection* y *Subtraction* mantienen la dependencia entre ellos. Como novedad, aquí no se limpian las memorias internas ya que es el módulo *Brightnes AD* el encargado de leer la imagen sustraída generada por el módulo *Subtraction* en la última iteración. Este hecho es posible debido a que el número de iteraciones es conocido de antemano (p veces).

IV. Resultados experimentales

Aunque existe una gran variedad de dispositivos y tecnologías FPGA, la arquitectura propuesta se ha implementado en un dispositivo FPGA ZC7Z020-CLG484 (arquitectura Artix) debido a su bajo coste y a su bajo consumo de energía, características que la hacen atractivs para su uso en aplicaciones embebidas. Cabe destacar que estos dispositivos poseen una relación equilibrada entre rendimiento y consumo energético. Sin embargo, los recursos de la FPGA en este tipo de dispositivos son limitados, lo que plantea un nuevo reto en el proceso de diseño de hardware, donde la replicación de módulos no es una solución factible.

La tabla I resume los recursos requeridos por cada configuración del acelerador hardware, que han sido extraídos de los informes post-síntesis. La arquitectura propuesta establece el tamaño del bloque hiperespectral en 1024 píxeles hiperespectrales (BS) con 12 bits de profundidad y 160 bandas, mientras que el número de elementos de procesamiento (PE) es variable y depende del número de bandas que contenga el bloque hiperespectral. El parámetro PE determina el número de bandas hiperespectrales que pueden ser procesadas simultáneamente por los módulos *Brightness*, *Brightness AD*, *Projection* y *Subtraction*. PE debe ser un divisor del número de bandas para que el modelo HLS pueda aplicar las optimizaciones de forma correcta. Cabe mencionar que los recursos disponibles del dispositivo admiten una configuración máxima de 20 PEs ya que la versión de 32 PEs requiere 262 DSPs, pero sólo se dispone de 220. Por tanto, los DSPs son el factor limitante que dificulta la escalabilidad de la arquitectura junto con el árbol de operaciones [15]. En cambio, los recursos BRAM no dependen del número de PEs, sino que varían en función del tamaño de los bloques (BS). Por lo tanto, todas las configuraciones utilizan aproximadamente el 70 % de este tipo des recursos, mientras que los FlipFlops y los LUTS no son críticos.

Los resultados de rendimiento se han medido como el número de fotogramas que se pueden procesar por segundo (FPS). La tabla I también enumera los FPS alcanzados por el acelerador según el número de PEs instanciados y la configuración de la frecuencia de reloj. Los resultados de rendimiento dibujan un comportamiento lineal, es decir, el rendimiento escala en consecuencia con el número de PEs. Además, las versiones más rápidas (143MHz) consiguen mejores resultados de rendimiento, aproximadamente un 40 % más que las lentas (100MHz). Cabe mencionar que una frecuencia de reloj superior, como 200MHz, no es posible porque las herramientas de síntesis no son capaces de cumplir las restricciones de temporización para esta arquitectura. Teniendo en cuenta las características de la cámara hiperespectral empleada (*Specim FX10*), que tiene una velocidad máxima de fotogramas de 327 FPS, siendo cada fotograma o línea capturada de 1024 píxeles con 224 bandas (rango completo) o/y 514 FPS para 1024 píxeles y 140 bandas por píxel [20], la versión que instancia 4 PEs con un reloj de 143MHz es la configuración más pequeña que puede ser utilizada con dicho sensor.

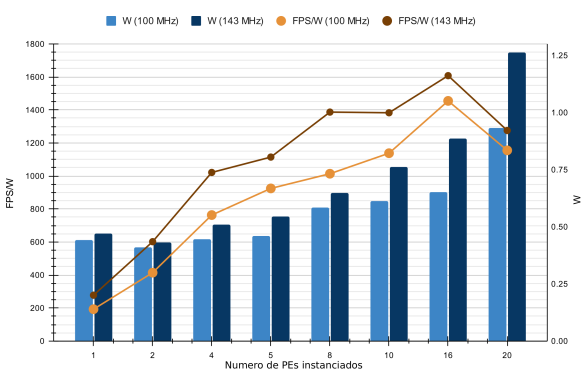

Fig. 5: Consumo de energía de las diferentes versiones del acelerador HW-LbL-FAD.

La figura 5 muestra gráficamente el consumo de energía obtenido por la implementación basada en FPGA para las diferentes versiones del acelerador en un dispositivo ZynQ-7020. Desde el punto de vista del consumo energético, la arquitectura propuesta es altamente eficiente (eje vertical derecho), oscilando entre 0,5 vatios, cuando se instancia un PE, y 1 y 1,3 vatios, cuando la arquitectura instancia 20 PEs con relojes de 100MHz y 143MHz, respectivamente. El consumo de los datos detectados dibuja un com-

Tabla II: Tamaño de los conjuntos de datos utilizados por las diferentes propuestas.

Propuesta	Dataset	Ejes espaciales Y	Ejes espaciales X	Eje espectral *(bandas)*
[21]	HyMap	614	512	126
	WTC	614	512	224
[22]	HyMap	100	300	126
	Hydice Forest	64	64	169
	Hydice Urban	80	100	175
[23]	San Diego	100	100	189
	Urban-Beach 1	100	100	207
	Urban-Beach 2	100	100	191
	Urban-Beach 3	100	100	205
	Urban-Beach 4	150	150	188
	EI Segundo	250	300	224
HW-LbL-FAD	Drone	825	1024	160

portamiento exponencial en lugar de lineal, debido al número de recursos de hardware y regiones de reloj que hay que alimentar.

Por otro lado, la eficiencia energética en términos de FPS por vatio también se muestra en la Figura 5 (series marrón y amarilla). La información que dibuja esta figura es demasiado relevante para las aplicaciones con limitaciones de presupuesto de energía, como las aplicaciones integradas en UAVs para el procesamiento en tiempo real a bordo. En este tipo de aplicaciones, es fundamental alargar la vida de la batería debido a escenarios desconocidos o no planificados, en los que es necesario un consumo extra para continuar la misión. Por ejemplo, una ráfaga de viento inesperada supone estabilizar el dron aumentando la velocidad de sus motores, y por tanto, el consumo de energía es mayor que en un escenario normal. En este sentido, esta figura de mérito puede ayudar a programar una misión con la garantía de completarla con éxito. Aunque la implementación a la frecuencia de reloj más alta es ligeramente mayor que la más lenta, se compensa con el número de FPS procesados. Por tanto, la versión más rápida (143MHz) tiene mejor relación FPS/W.

V. Discusión

El rendimiento y la utilización de recursos hardware de la arquitectura presentada en este trabajo para el algoritmo HW-LbL-FAD se ha comparado con tres detectores de anomalías del estado del arte, que han sido implementados en lógica reconfigurable, en concreto en dispositivos de Xilinx de la serie 7. B. Yang et al. han optimizado el algoritmo RX mediante un enfoque estadístico del fondo, donde han elegido una FPGA Kintex-7 XC7K325T para su prototipado [21]. Otro algoritmo de detección de anomalías interesante es el Detector Basado en la Representación Colaborativa (CRD), que se ha sido implementado por W. Jingjing et al. en una FPGA Virtex-7 XC7VX980T [22]. Por último, el detector de anomalías presentado por J. Lei et al. en [23] se basa en la reconstrucción morfológica y un filtro guiado simplificado (Fast-MGD) que se ha implementado en una Virtex-7 XC7VX690T.

Cada uno de los detectores de anomalías del estado del arte mencionados anteriormente utiliza su propio conjunto de datos, que se enumeran en la Tabla II. El tamaño de cada cubo hiperespectral se define por el número de líneas (eje espacial Y), el tamaño del bloque (eje espacial X) y el número de bandas (eje espectral). Debido a la variedad de conjuntos de datos, la comparación temporal se ha realizado por el número de elementos dentro del cubo hiperespectral independientemente del detector de anomalías, definido por la multiplicación de los dos ejes espaciales y el espectral. La figura 6 muestra gráficamente el tiempo (eje vertical derecho, barras azul claro) necesario para calcular el número de elementos del cubo hiperespectral (eje vertical izquierdo, barras azul oscuro). Además, en la figura 6 también se dibuja el número de elementos procesados por segundo (eje vertical derecho, línea naranja). Cabe destacar que la arquitectura propuesta tiene un mejor rendimiento temporal que las otras propuestas; el detector HW-LbL-FAD procesa hasta cinco y siete veces más datos que la segunda mejor propuesta ([21]) en sus versiones más lenta (100MHz) y más rápida (143MHz), respectivamente. Aunque las propuestas presentadas en [22] y [23] emplean cubos hiperspectrales muy pequeños, ambas consiguen un peor equilibrio entre tiempo y rendimiento.

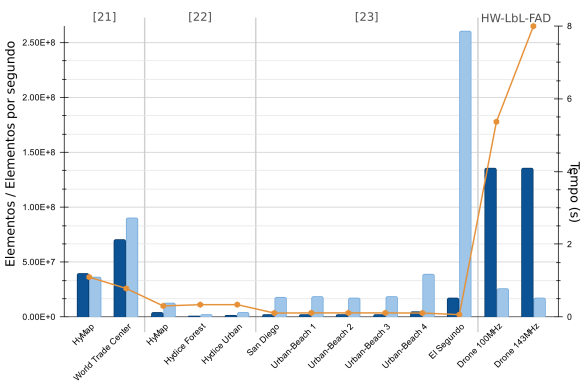

Fig. 6: Tiempo de procesado de elementos de los diferentes detectores de anomalías.

Desde el punto de vista de la utilización hardware, la Tabla III muestra la utilización de recursos por cada detector de anomalías y su porcentaje según el dispositivo empleado y el conjunto de datos con el que se ha trabajado. Tal vez, esta tabla induzca a error al lector al tratar de interpretarla, ya que se puede observar que los detectores del estado del arte utilizan un pequeño porcentaje de los recursos hardware disponibles. Sin embargo, no es así, las propuestas del estado del arte han seleccionado dispositivos FPGA de altas prestaciones que hacen que sus resultados parezcan muy prometedores, pero el hecho es que la mayoría de las implementaciones no caben en una FPGA de coste optimizado. El uso de dispositivos de gama media y grande (arquitecturas Kintex y Virtex, respectivamente) en lugar de los de coste optimizado (arquitectura Artix) introduce una nueva dimensión en el proyecto: el coste del producto final. La tabla III también muestra el precio de cada dispositivo FPGA en la última columna. Las arquitecturas basadas en dispositivos Virtex-7, [22] y [23], tienen un coste económico elevado que puede ser inviable en sistemas embebidos, mientras que

Tabla III: Recursos hardware utilizados por los detectores del estado del arte y por la arquitectura presentada (20PEs).

	Dispositivo	**Dataset**	**BRAM18K**	**DSP48E**	**FlipFlops**	**LUTs**	**Frec.** (MHz)	**Consumo** (W)	**Coste (€)**
[21]	XC7VX690T	HyMap	240 *(26,97 %)*	265 *(31,55 %)*	28.245 *(6,93 %)*	21.730 *(10,66 %)*	200	0,641	30.949,10
		WTC	284 *(9,47 %)*	459 *(12,75 %)*	43.544 *(3,56 %)*	33.997 *(5,56 %)*	200	0,897	
[22]	XC7VX980T	HyMap	4 *(0,13 %)*	1.064 *(29,56 %)*	772.164 *(63,09 %)*	269.751 *(44,08 %)*	100	2,72	16.577,93
		H. Forest	3 *(0,10 %)*	460 *(12,78 %)*	277.048 *(22,63 %)*	140.362 *(22,93 %)*	100	1,333	
		H. Urban	3 *(0,10 %)*	580 *(16,11 %)*	414.673 *(33,88 %)*	146.977 *(24,02 %)*	100	1,619	
[23]	XC7K325T	San Diego	216 *(7,35 %)*	12 *(0,33 %)*	49.964 *(5,77 %)*	33.969 *(7,84 %)*	200	0,614	2.103,98
		U.-Beach 1	282 *(9,59 %)*	12 *(0,33 %)*	57.962 *(6,69 %)*	43.890 *(10,13 %)*	200	0,687	
		U.-Beach 2	198 *(6,73 %)*	12 *(0,33 %)*	46.370 *(5,35 %)*	31.681 *(7,31 %)*	200	0,593	
		U.-Beach 3	282 *(9,59 %)*	12 *(0,33 %)*	57.929 *(6,69 %)*	43.874 *(10,13 %)*	200	0,686	
		U.-Beach 4	282 *(9,59 %)*	12 *(0,33 %)*	58.593 *(6,76 %)*	44.906 *(10,37 %)*	200	0,691	
		El Segundo	525 *(17,86 %)*	12 *(0,33 %)*	83.713 *(9,66 %)*	67.591 *(15,60 %)*	200	0,911	
HW-LbL -FAD	ZC7Z020	Drone (*propio*)	197 *(70,36 %)*	166 *(75,45 %)*	23.071 *(21,68 %)*	23.493 *(44,16 %)*	100 143	0,7 1,077	154,7

la basada en un dispositivo Kintex-7 ([21]) tiene un precio desorbitado, aunque sus prestaciones pueden llegar a procesar grandes cubos hiperespectrales. Sin embargo, nuestra propuesta es una solución de bajo coste que explota al máximo la tecnología FPGA, obteniendo el mejor compromiso entre rendimiento y coste.

Desde el punto de vista energético, la Tabla III recoge en su penúltima columna la potencia en el chip representada en vatios, que ha sido estimada con la herramienta Xilinx Power Estimator (XPE) debido a que las citadas propuestas no proporcionan esta información. En aras de la claridad, la Figura 7 dibuja los elementos por segundo (barras verde oscuro) y los elementos procesados por vatio (barras verde claro). Podemos concluir que nuestra solución supera a las demás propuestas del estado del arte en el balance de potencia, donde la versión más lenta y la más rápida de la arquitectura HW-LbL-FAD son entre 5 y 107 veces mejores que los detectores de anomalías presentados en la literatura. Cabe destacar que ambas versiones del detector HW-LbL-FAD tienen un número similar de elementos procesados por vatio (ver Figura 7).

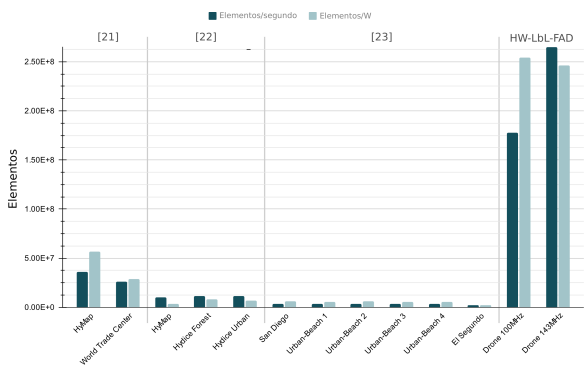

Fig. 7: Relación de consumo de potencia de los detectores de anomalías.

VI. Conclusiones

El principal reto al que se ha enfrentado este trabajo ha sido el diseño de una solución de bajo consumo y eficiente desde el punto de vista energético para un algoritmo de detección de anomalías implementado sobre una plataforma de lógica reconfigurable. Además de las limitaciones de recursos y energía, también se atendieron los requisitos de alto rendimiento.

La implementación inicial del conjunto de operaciones hiperespectrales es heredada de trabajos anteriores [15], [16] donde se comprobó la idoneidad de la tecnología FPGA para este tipo de aplicaciones. Así pues, esta estrategia ha ahorrado una cantidad importante de tiempo y esfuerzo, permitiendo a los autores centrarse en la mejora de las prestaciones y en el diseño de un *datapath* y una lógica de control que permitiera compartir el tiempo de los módulos hardware y el uso óptimo de los recursos. También se ha ido un paso más allá y se ha proporcionado modificaciones funcionales mínimas a los operadores para que sean versátiles y se puedan reutilizar fácilmente durante las diferentes etapas del algoritmo. Como resultado, el conjunto básico de operadores hiperespectrales hardware no sólo son válidos para la compresión de imágenes [15] sino para la tarea de detección de anomalías, sin incurrir en ninguna sobrecarga.

En términos de rendimiento de detección, la arquitectura diseñada para el algortimo HW-LbL-FAD alcanza la misma precisión que su versión software. Esto significa que la precisión de la detección de píxeles anómalos es alta y mayor que la alcanzada por los algoritmos del estado del arte. En cuanto a la compensación de tiempo de procesamiento, la arquitectura propuesta procesa grandes cubos hiperespectrales en un corto periodo de tiempo con un pequeño presupuesto de energía en comparación con las propuestas del estado del arte. Además, el coste de producción de nuestra arquitectura, al utilizar dispositivos FPGA, es muy atractivo debido al bajo coste del dispositivo utilizado.

Agradecimientos

Esta investigación está parcialmente financiada por el Ministerio de Economía y Competitividad (MINECO) del Gobierno de España (proyecto TALENT, no. PID2020-116417RB-C4, subproyectos 1 y 4), la Agencia Canaria de Investigación, Innovación y Sociedad de la Información (ACIISI) de la Conserjería de Economía, Industria, Comercio y Conocimiento del Gobierno de Canarias, conjuntamente con el Fondo Social Europeo (FSE) (POC2014-2020, Eje 3 Tema Prioritario 74 (85 %)), y por el programa Europeo Horizonte 2020 bajo el proyecto SHAPES (GA Nº 857159).

Referencias

[1] Hongjun Su, Zhaoyue Wu, Huihui Zhang, and Qian Du, "Hyperspectral anomaly detection: A survey," *IEEE Geoscience and Remote Sensing Magazine*, 2021.

[2] Sukanta Roy, Sumit Pathak, and Subbaramajois Narasipur Omkar, "Automated mineralogical anomaly detection using a categorization of optical maturity trend at lunar surface," *International Journal of Remote Sensing*, vol. 42, no. 21, pp. 8262–8297, 2021.

[3] Jannick Kuester, Wolfgang Gross, and Wolfgang Middelmann, "Hyperspectral anomaly detection of hidden camouflage objects based on convolutional autoencoder," in *Artificial Intelligence and Machine Learning in Defense Applications III*. SPIE, 2021, vol. 11870, pp. 98–106.

[4] Irving S Reed and Xiaoli Yu, "Adaptive multiple-band cfar detection of an optical pattern with unknown spectral distribution," *IEEE transactions on acoustics, speech, and signal processing*, vol. 38, no. 10, pp. 1760–1770, 1990.

[5] Yang Xu, Zebin Wu, Jun Li, Antonio Plaza, and Zhihui Wei, "Anomaly detection in hyperspectral images based on low-rank and sparse representation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 4, pp. 1990–2000, 2016.

[6] Yuxiang Zhang, Bo Du, Liangpei Zhang, and Shugen Wang, "A low-rank and sparse matrix decomposition-based mahalanobis distance method for hyperspectral anomaly detection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 3, pp. 1376–1389, 2016.

[7] Jiayi Li, Hongyan Zhang, Liangpei Zhang, and Li Ma, "Hyperspectral anomaly detection by the use of background joint sparse representation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2523–2533, 2015.

[8] Wei Li and Qian Du, "Collaborative representation for hyperspectral anomaly detection," *IEEE Transactions on geoscience and remote sensing*, vol. 53, no. 3, pp. 1463–1474, 2015.

[9] Fei Li, Xiuwei Zhang, Lei Zhang, Dongmei Jiang, and Yanning Zhang, "Exploiting structured sparsity for hyperspectral anomaly detection," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 7, pp. 4050–4064, 2018.

[10] Pangambam Sendash Singh and Subbiah Karthikeyan, "Salient object detection in hyperspectral images using deep background reconstruction based anomaly detection," *Remote Sensing Letters*, vol. 13, no. 2, pp. 184–195, 2022.

[11] Mahdi Yousefan, Hamid Esmaeili Najafabadi, Hossein Amirkhani, Henry Leung, and Vahid Hajihashemi, "Deep anomaly detection in hyperspectral images based on membership maps and object area filtering," *Expert Systems with Applications*, p. 116200, 2021.

[12] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki, "The singular value decomposition: Anatomy of optimizing an algorithm for extreme scale," *SIAM Review*, vol. 60, no. 4, pp. 808–865, 2018.

[13] Adrián Alcolea, Mercedes E Paoletti, Juan M Haut, Javier Resano, and Antonio Plaza, "Inference in supervised spectral classifiers for on-board hyperspectral imaging: An overview," *Remote Sensing*, vol. 12, no. 3, pp. 534, 2020.

[14] Alan D George and Christopher M Wilson, "Onboard processing with hybrid and reconfigurable computing on small satellites," *Proceedings of the IEEE*, vol. 106, no. 3, pp. 458–470, 2018.

[15] Julián Caba, María Díaz, Jesús Barba, Raúl Guerra, and Jose A. de la Torre and Sebastián López, "Fpga-based on-board hyperspectral imaging compression: Benchmarking performance and energy efficiency against gpu implementations," *Remote Sensing*, vol. 12, no. 22, 2020.

[16] Raúl Guerra, Yubal Barrios, María Díaz, Abelardo Baez, Sebastián López, and Roberto Sarmiento, "A hardware-friendly hyperspectral lossy compressor for next-generation space-grade field programmable gate arrays," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 12, pp. 4813–4828, 2019.

[17] Erick L Oberstar, "Fixed-point representation & fractional math," *Oberstar Consulting*, p. 9, 2007.

[18] Chuanmin Hu, Lian Feng, Zhongping Lee, Curtiss Davis, Antonio Mannino, Charles Mcclain, and Bryan Franz, "Dynamic range and sensitivity requirements of satellite ocean color sensors: Learning from the past," *Applied optics*, vol. 51, pp. 6045–62, 09 2012.

[19] Ashok Kumar, Sanjeev Mehta, Sandip Paul, R. Parmar, and R Samudraiah, "Dynamic range enhancement of remote sensing electro-optical imaging systems," 12 2012.

[20] Specim, Spectral Imaging Ltd, "Specim fx10 datasheet," Available Online: `https://www.specim.fi/wp-content/uploads/2020/02/Specim-FX10-Technical-Datasheet-04.pdf`, (Último acceso: 12 de mayo de 2022).

[21] Bin Yang, Minhua Yang, Antonio Plaza, Lianru Gao, and Bing Zhang, "Dual-mode fpga implementation of target and anomaly detection algorithms for real-time hyperspectral imaging," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2950–2961, 2015.

[22] Jingjing Wu, Yu Jin, Wei Li, Lianru Gao, and Bing Zhang, "Fpga implementation of collaborative representation algorithm for real-time hyperspectral target detection," vol. 15, no. 3, pp. 673–685, oct 2018.

[23] Jie Lei, Geng Yang, Weiying Xie, Yunsong Li, and Xiuping Jia, "A low-complexity hyperspectral anomaly detection algorithm and its fpga implementation," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 907–921, 2021.