TRABAJO FIN DE MÁSTER

MODELADO DEL COMPRESOR DE IMÁGENES MULTIESPECTRALES E HIPERESPECTRALES CCSDS-123 MEDIANTE COFLUENT STUDIO

Titulación: Máster en Tecnologías de Telecomunicación

Autor: Ana Gómez Rebordinos

Tutores: Dr. Sebastián López Suárez
Dr. Roberto Sarmiento Rodríguez
Dra. Lucana Santos Falcón

UNIVERSIDAD DE 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

MODELADO DEL COMPRESOR DE IMÁGENES MULTIESPECTRALES E HIPERESPECTRALES CCSDS 123 MEDIANTE COFLUENT STUDIO

Autor: Ana Gómez Rebordinos
Tutor(es): D. Sebastián López Suárez
D. Roberto Sarmiento Rodríguez
Dña. Lucana Santos Falcón
Fecha: Julio 2015

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

Versión: 2.0 – 14 junio 2011

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

Máster en
Tecnologías de Telecomunicación

IUMA

Trabajo Fin de Máster

MODELADO DEL COMPRESOR DE IMÁGENES MULTIESPECTRALES E HIPERESPECTRALES CCSDS 123 MEDIANTE COFLUENT STUDIO

HOJA DE FIRMAS

Alumno/a:	Ana Gómez Rebordinos	Fdo.:
Tutor/a:	D. Sebastián López Suárez	Fdo.:
Tutor/a:	D. Roberto Sarmiento Rodríguez	Fdo.:
Tutor/a:	Dña. Lucana Santos Falcón	Fdo.:

Fecha: Julio 2015

IUMA
t +34 928 451 086 | iuma@iuma.ulpgc.es
f +34 928 451 083 | 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

MODELADO DEL COMPRESOR DE IMÁGENES MULTIESPECTRALES E HIPERESPECTRALES CCSDS 123 MEDIANTE COFLUENT STUDIO

HOJA DE EVALUACIÓN

Calificación: .....

Presidente	Javier A. García García	Fdo.:
Secretario	Francisco Javier del Pino Suárez	Fdo.:
Vocal	Félix B. Tobajas Guerrero	Fdo.:

Fecha: Septiembre 2015

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

Versión: 2.0 – 14 junio 2011

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. Introducción

1.1. Resumen

Las imágenes multiespectrales e hiperespectrales son imágenes digitales tomadas por sensores capaces de captar información en diferentes rangos de longitudes de onda. Cada píxel se corresponde con un vector de tantas componentes como longitudes de onda sea capaz de distinguir el sensor, asociadas cada una de ellas a una banda de la imagen multiespectral o hiperespectral. Como resultado, la imagen hiperespectral supone un volumen de datos de tamaño considerable.

A partir de la información contenida en las imágenes hiperespectrales es posible aplicar diferentes técnicas de procesado, tales como el desmezclado hiperespectral, que permite diferenciar de manera unívoca los materiales presentes en la escena y estimar la abundancia de cada uno de ellos; técnicas de clasificación, de detección, identificación, y muchos más procesos clave para la investigación y la ciencia. Este tipo de imágenes representan una de las fuentes de información más potentes dentro del campo denominado teledetección o *remote sensing*, y conllevan, con el aumento de la resolución de los sensores, un aumento de la cantidad de datos a procesar. La alta dimensionalidad de las imágenes hiperespectrales, la limitación de recursos y de cómputo, así como las capacidades de transmisión de los sistemas en el espacio, que son los de interés para este Trabajo Fin de Máster, hacen necesaria una reducción de la cantidad de información a través de la compresión de datos.

Uno de los algoritmos de compresión desarrollados con dicho propósito es el CCSDS-123, *Consultative Committee for Space Space Data System*-123 [1], que consta de una primera etapa de predicción y una posterior de codificación entrópica, como se muestra en la Fig. 1, y será el objeto de estudio de este Trabajo Fin de Máster, cuyo objetivo principal es modelar y simular el citado algoritmo, analizando sus prestaciones, de modo que la información obtenida sirva de base para tomar decisiones arquitecturales en una futura implementación a bajo nivel sobre una FPGA o un ASIC.

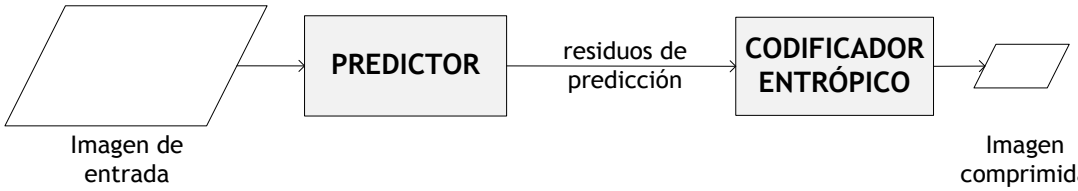

Fig. 1: Esquema del compresor CCSDS 123.

Para realizar el modelado se utiliza la herramienta software Intel CoFluent Studio [2][3][4][5] que permite modelar y simular sistemas electrónicos, y realizar una exploración del espacio de diseño a partir de dichos modelos.

El uso de esta herramienta mejora el ciclo de vida del desarrollo de sistemas, facilitando el proceso de elección de arquitectura y la definición de distintos escenarios para la validación del comportamiento en tiempo real, prediciendo rendimiento y generando casos de test para la implementación.

1.2. Abstract

Multispectral and hyperspectral images are digital images taken by sensors capable of capturing information in different wavelength ranges. In such an image, each pixel corresponds to a so-many-components vector as different wavelengths the sensor is able to distinguish. The resulting image is equivalent to a data cube of considerable size.

Several processes can be performed based on the information present in the hyperspectral images, such as hyperspectral unmixing, differentiating materials contained in the scene and the abundance of each one of them; classification processes, detection, identification and many other key processes for current research and science. This kind of images represents one of the most powerful tools in the field known as remote sensing, and the amount of associated data tends to increase, as the resolutions of the sensors also grows.

Specific conditions of the present systems in space systems for remote sensing along with specific characteristics of hyperspectral images lead to a storage and bandwidth limitation and thus reducing the amount of information by means of compression is essential.

The CCSDS-123 algorithm is a standard compressor specifically developed

to reduce the data volume of hyperspectral images collected on-board satellites. It is based on a prediction stage followed by an entropy coding process, as shown in Fig. 1.

The main objective of this work is to model the CCSDS-123, simulate it and analyze its performance, in such a way that the obtained metrics aid taking architectural decisions during a possible subsequent implementation phase on an FPGA or ASIC.

In particular we will employ the Intel CoFluent Studio software tool, which is useful for modelling and simulating electronic systems and performs design space exploration, improving life cycle of systems development, by aiding architecture election and defining different scenarios for validation and verification.

1.3. Objetivos del TFM

El objetivo principal de este Trabajo Fin de Máster, enmarcado en el proyecto ITT No. AO/1-8032/14/NL/AK – CCSDS Lossless Compression IP-Core Space applications [6] del programa TRP-ESA, es el modelado y simulación de un sistema de compresión de imágenes multiespectrales e hiperespectrales que cumpla con el estándar CCSDS-123.

Considerando el proceso de compresión de imágenes descrito en dicho estándar, se realizará un estudio del sistema, formado por la etapa de predicción y la etapa de codificación entrópica. De esta última, se estudiarán las dos opciones admitidas por el estándar: el codificador entrópico *sample-adaptive*, y el codificador *block-adaptive* correspondiente al estándar CCSDS-121 [7]. Para dicho estudio se partirá del software de referencia proporcionado por la ESA, *European Space Agency*.

2. Estado del arte

2.1. Adquisición de imágenes hiperespectrales

2.1.1. Imágenes multiespectrales e hiperespectrales

Las imágenes multiespectrales e hiperespectrales consisten en una determinada cantidad de información adquirida a lo largo de un rango de longitudes de onda del espectro electromagnético. Se puede entender como un conjunto de imágenes del mismo objeto o escena, representadas cada una de ellas en diferentes longitudes de ondas. Por lo general, los datos adquiridos se organizan en forma de un cubo como el que se ilustra en Fig. 2.

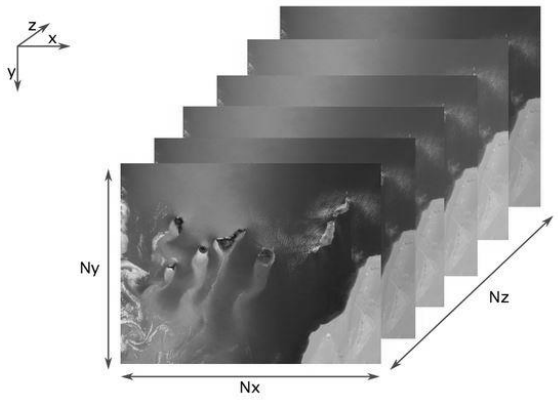

Fig. 2: Cubo de datos hiperespectral

Cada muestra de la imagen se corresponde con un pixel que a su vez, se corresponde con un vector de tantas componentes como bandas haya en el cubo. La información espacial es la contenida en los distintos planos X-Y y relativa a $N_x \times N_y$ píxeles, mientras que la información espectral es la contenida a lo largo del eje Z, relativa a N_z bandas espectrales.

La principal diferencia entre imágenes multiespectrales e hiperespectrales se puede atribuir esencialmente al número de bandas por las que están compuestas y a la continuidad de esas bandas. Típicamente una imagen multiespectral está formada por un orden de decenas de bandas no necesariamente contiguas unas a otras, mientras que una imagen hiperespectral cuenta con un número bastante mayor de bandas contiguas entre ellas, de modo que se puede equiparar dicha imagen a un espectro

continuo o a lo que se conoce como firma espectral del objeto o escena de análisis. Debido a que las técnicas de procesado presentadas en este TFM se pueden aplicar indistintamente a ambos tipos de imagen, a lo largo de este documento, se utilizará el término hiperespectral tanto para designar a las imágenes multiespectrales como hiperespectrales.

2.1.2. Instrumentación en imágenes hiperespectrales

Los instrumentos denominados espectrómetros de imágenes son los que permiten obtener las mencionadas imágenes hiperespectrales. Estos instrumentos o sensores hiperespectrales permiten la obtención de la información detallada o firma espectral para cada pixel de la imagen, que se obtiene capturando la reflectancia por el propio sensor en diferentes longitudes de onda.

Los sensores se pueden clasificar de varias formas. Teniendo en cuenta el diseño de adquisición, y en concreto su geometría a la hora de la captura de la imagen, se encuentran los siguientes tipos de sensores, tal y como se ilustra en los ejemplos de Fig. 3:

- Sensores de barrido o rotación (*whiskbroom*).

  La captura de la imagen se realiza mediante un proceso continuo, y lleva asociado un único sensor con un prisma o espejo con capacidad de rotación para cambiar el punto a capturar. Los satélites LANDSAT previos al LANDSAT 8 han contado con este tipo de diseño, así como los NOAA y GOES, entre otros.

- Sensores lineales o de empuje (*pushbroom*).

  Consiste en una serie de sensores alineados entre ellos que capturan la imagen mediante un proceso continuo y línea a línea, y por tanto cada uno de ellos presenta sus propios parámetros de orientación. Ejemplos de sistemas con este tipo de diseño son LANDSAT 8, IRS, y PLEIADES, entre otros.

- Sensores matriciales (*frame cameras*).

  Se trata de un conjunto de sensores que se disponen en el plano

focal de forma matricial, permitiendo capturar la imagen en un solo instante.

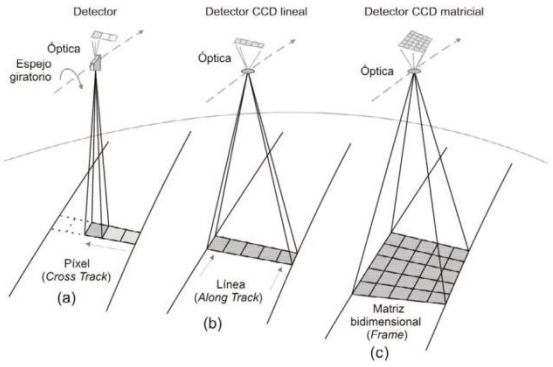

Fig. 3: Sensores hiperespectrales remotos

(a) Sensores barrido (b) Sensores de empuje (c) Sensores matriciales

La elección de un tipo de sensor u otro depende básicamente del fin y requerimientos de la aplicación en concreto. Una vez elegido el sensor, incorporado al sistema y adquirida la imagen, la información que se almacena y/o trata a continuación se encuentra organizada de una forma específica y con unas características concretas, que se explicarán más adelante.

2.1.3. Aplicaciones de imágenes hiperespectrales

Desde las primeras capturas de imágenes hiperespectrales, su análisis ha sido un área bastante activa en la ciencia y la investigación, aunque su avance se ha visto limitado principalmente por la poca disponibilidad de dichas imágenes. Sin embargo, tras la aparición de sistemas comerciales con este tipo de tecnología, se han incrementado significativamente sus aplicaciones y los campos de dichas aplicaciones, entre los que destacan:

- Investigación forense: verificación de documentos, identificación de materiales...
- Medicina: diagnóstico no invasivo, biopsias, delimitación de tejidos...
- Explotaciones mineras: análisis de perforaciones, mapeado de minerales, análisis de canteras y excavaciones…
- Detección remota: detección de contaminantes, cartografía...

- Investigación agrícola: inspección en líneas de producción, gestión y monitorización de cultivos, calidad de alimentos...
- Ingeniería química, biología y biotecnología: fluorescencia, laboratorios de I+D, detección de nanopartículas...
- Militar y defensa: protección de fronteras, reconocimiento de territorios...
- Procesos industriales: control de calidad, materiales semiconductores, colorimetría, control de procesos en fabricación…
- Aeroespacial: condiciones atmosféricas, satélites...

2.2. Compresión en imágenes hiperespectrales

2.2.1. Compresión de imágenes hiperespectrales

Tanto por el tipo como por la cantidad de información contenida en las imágenes hiperespectrales, son bastante evidentes las ventajas y el gran potencial de este tipo de imágenes. Sin embargo, este potencial viene acompañado también de importantes retos: en determinados sistemas cuyas capacidades computacionales están limitadas, tanto el tratamiento como el almacenamiento de tal cantidad de información, del orden de gigaBytes, pueden suponer un problema.

En el caso concreto de detección remota o teledetección, que es la aplicación en que se centra este Trabajo Fin de Máster, nos enfrentamos a distintos problemas por la alta dimensionalidad de los datos.

Primeramente, al tratarse de sistemas embarcados en aviones o satélites, la capacidad computacional para el tratamiento de las cientos de bandas de cada una de las imágenes obtenidas, así como lo está la capacidad de almacenamiento, están limitadas.. Por otro lado, una vez tratada la información o directamente desde su recopilación y en algún momento futuro, dicha información debe ser enviada desde el satélite desde el que se ha obtenido hasta alguna estación terrena, por lo que pueden darse también

problemas asociados a los límites en el ancho de banda de la transmisión.

Por ende, tanto por la economía de dicha transmisión, entendida como la reducción del ancho de banda o de los tiempos de transmisión, como por la reducción de la cantidad de información que se debe almacenar, la compresión de las imágenes hiperespectrales suele ser necesaria en este tipo de sistemas. La compresión se fundamenta en representar la información necesaria con la menor cantidad posible de bits. La compresión de imágenes hiperespectrales permite reducir de manera eficiente el volumen de datos por los motivos que se explican a continuación. Los píxeles en estas imágenes están muy correlados, tanto a nivel espacial como a nivel espectral, y esto permite reducir significativamente el tamaño eliminando esa redundancia. Igualmente, existe redundancia en los símbolos o información a transmitir, en el sentido de que hay algunos símbolos más probables que otros, de modo que es posible reducir el volumen de los datos seleccionando adecuadamente los códigos de palabra asociados a cada símbolo.

En cuanto a las técnicas empleadas para la compresión de imágenes, existen dos principales categorías: compresión con pérdidas y compresión sin pérdidas. En la primera categoría, compresión con pérdidas, se consiguen tasas de compresión mayores, a través del descarte de cierta información, lo que hace imposible recuperar con total fidelidad la imagen original; mientras que en la segunda categoría se utilizan algoritmos que consiguen reducir el volumen de información a almacenar o transmitir, con la premisa de que al descomprimir dicha información se obtenga exactamente la misma imagen que se tenía antes de la compresión.

A continuación se exponen datos referentes a varias misiones espaciales, con el fin de recalcar la importancia y necesidad de este proceso.

La misión *Euclid*, con lanzamiento previsto en 2020, pretende explorar la materia oscura y ayudar a entender el origen de la expansión del Universo. Para ello contará con un espectrómetro de infrarrojo cercano y una cámara de alta resolución en el óptico. Con este sistema se estima obtener alrededor de 1.08 terabits de información diaria, a través de imágenes de unos 10 gigabits; sin embargo, el sistema está limitado a 520 gigabits diarios. Por tanto, es

necesario un proceso de compresión, y está determinado que sea compresión sin pérdidas.

La misión *HyspIRI* de la NASA, contempla el estudio de los ecosistemas y la obtención de información crítica en desastres naturales, tales como volcanes. Para tal fin cuenta con dos instrumentos a bordo de un satélite polar: un espectrómetro que abarca desde el espectro visible hasta el infrarrojo cercano y un sensor multiespectral para el infrarrojo medio y térmico. La cantidad de información que se obtiene diariamente asciende a cinco terabits, mientras que la capacidad de almacenamiento a bordo es de aproximadamente 200 gigabytes, por lo que se requiere igualmente de un proceso de compresión.

Por último, la misión *Sentinel* de la ESA, formada por varias parejas de satélites, pretende realizar una observación de La Tierra remplazando misiones anteriores y asegurando la continuidad de los datos para los estudios actuales. Cada parte de la misión se centrará en un aspecto determinado de la observación terrestre: atmósfera, océanos, monitorización terrestre… No es de extrañar que en esta misión, se dé la situación de las misiones nombradas anteriormente, en la que la cantidad de datos requieran una compresión en algún momento del proceso.

2.2.2. Algoritmos de compresión de imágenes hiperespectrales

Por lo general, los algoritmos de compresión de imágenes hiperespectrales constan de una etapa de decorrelación, espacial y/o espectral, una etapa de cuantificación y una etapa de codificación.

- Etapa de decorrelación

  Esta etapa consiste en la reducción de información relativa a la redundancia espacial y/o temporal a una mínima cantidad, de manera que en la descompresión de la imagen se pueda recuperar la original, o una aproximación si se trata de un algoritmo con pérdidas.

  Esta etapa puede estar basada en transformadas (Transformada Discreta del Wavelet, WDT; Transformada Discreta del Coseno, CDT; o

Transformada de Kahrunen-Loève, entre otros) en las cuales la codificación se limita a los coeficientes obtenidos; o basada en predicción (las muestras se estiman a través de la información de muestras vecinas a nivel espacial y/o a nivel espectral) y la codificación se aplica a los residuos o errores de dichas predicciones.

- Etapa de cuantificación

La etapa de cuantificación se encarga de asignar a cada error de predicción un número entero positivo adecuado para el codificador entrópico adaptativo.

- Etapa de codificación

Esta etapa es la que aborda la redundancia de la propia información a transmitir, en base a un determinado modelo de probabilidad. Este tipo de codificación se denomina codificación entrópica, y entre los códigos más comunes, se pueden encontrar: códigos de Huffman, códigos aritméticos, códigos universales o códigos de Golomb.

Con independencia del método de codificación entrópica seleccionado, existe un límite conocido como límite de Shannon, que establece el número mínimo de bits necesarios para la codificación de un conjunto de símbolos o eventos, y se calcula como se indica a continuación:

$$H(P) = \sum_{k=1}^{n} -p\{e_k\} \log_2 p\{e_k\} \qquad \text{(Ecuación 1)}$$

donde $p\{e_k\}$ es la probabilidad de que el evento o símbolo e_k ocurra.

La eficiencia de la compresión, como se podrá intuir por las características de las imágenes explicadas previamente, depende en cierta medida de la propia imagen a comprimir, pero también depende del método de compresión elegido; es por ello necesario establecer métricas que permitan comparar los distintos métodos y la eficiencia de estos.

El ratio de compresión es la métrica más común, y se expresa generalmente de una de las siguientes maneras:

- Ratio de compresión (*CR*, del inglés *Compression Ratio*)

$$CR = \frac{Tamaño\ de\ imagen\ original\ (bits)}{Tamaño\ de\ imagen\ comprimida\ (bits)} \qquad \text{(Ecuación 2)}$$

- Bits por pixel por banda (*bpppb,* del inglés *bits per pixel per band*)

$$bpppb = \frac{Tamaño\ de\ imagen\ comprimida\ (bits)}{Líneas\ x\ Columnas\ x\ Bandas} \qquad \text{(Ecuación 3)}$$

Se ha mencionado ya que los métodos de compresión con pérdidas permiten alcanzar mayores ratios de compresión que los ratios obtenidos con los métodos sin pérdidas, pero igualmente, que la imagen sólo puede ser correctamente descomprimida, en el sentido de poder recuperar exactamente la imagen original, a través de métodos sin pérdidas, por lo que la elección de un tipo u otro de compresor no es una decisión trivial, si bien existe cierta tendencia a los algoritmos sin pérdidas, al mantener toda la información de los cubos hiperespectrales.

También se encuentran en la literatura científica dos tipos de métodos de decorrelación para imágenes hiperespectrales: algunos algoritmos abordan el proceso desde un punto de vista bidimensional, considerando únicamente la redundancia espacial en la etapa de predicción, como LOCO-I [8] o 2D-CALIC [9]; otras propuestas sí tienen en consideración la redundancia entre bandas espectrales, consiguiendo unos ratios de compresión mayores al aprovechar dicha redundancia espectral. Ejemplos de este último tipo son LCL-3D [10] y 3D-CALIC [11].

Los ratios de compresión en métodos sin pérdidas están en torno a un factor 2 o 3, por la limitación impuesta al no ser posible el descarte de parte de información de las imágenes originales; sin embargo, la tendencia comentada anteriormente hacia estos métodos puede cambiar en un futuro cuando sea crítico alcanzar ratios de compresión cada vez mayores, para las nuevas generaciones de instrumentos de obtención de estas imágenes.

En la siguiente imagen, Fig. 4, se muestran los resultados de los compresores sin pérdidas JPEG-LS (Lossless Joint Photographic Experts Group) y una implementación del estándar CCSDS-123, conocida como Emporda, en cuanto a bits por pixel por banda, junto con los bits por pixel por banda de las imágenes sin comprimir (RAW), siendo ambos estándar. Pese a que el

resultado varía entre imágenes, a rasgos generales se observa una mejora general del CCSDS123-Emporda sobre el resto. Este hecho, junto con todo el trabajo desarrollado en torno a este estándar y los resultados obtenidos, como el realizado en [14] o su versión sin pérdidas [15], hacen del estándar CCSDS-123 un buen candidato a modelar, lo que permitirá confirmar la buena relación existente entre la eficiencia de compresión y la baja complejidad del algoritmo.

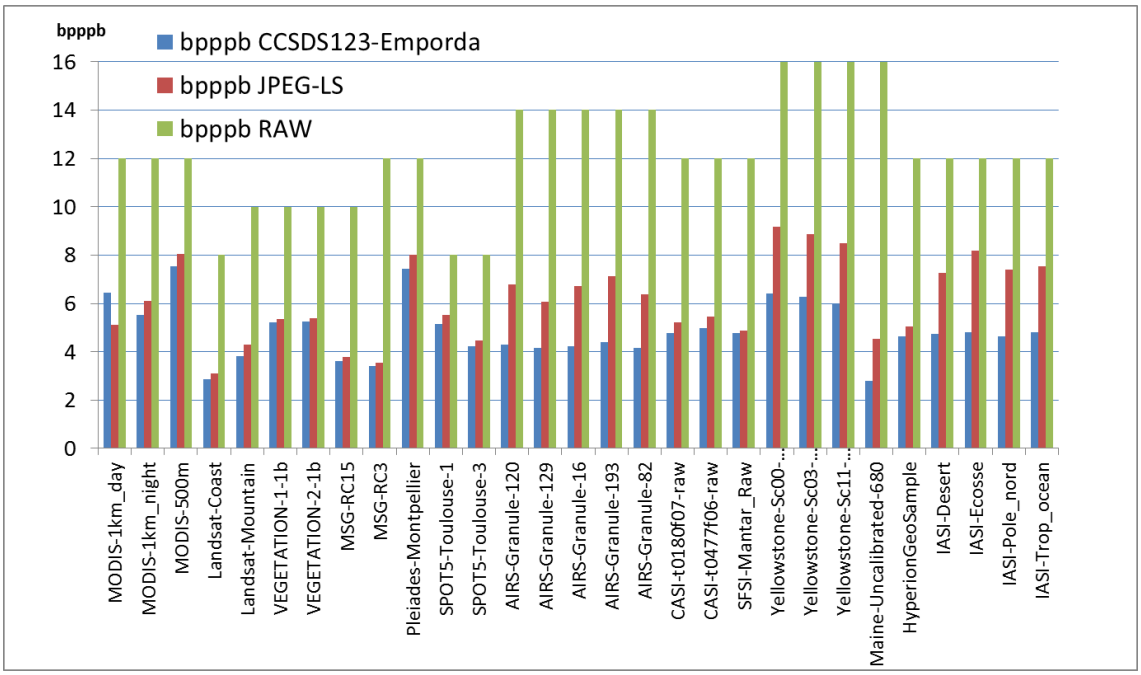

Fig. 4: Comparativa algoritmos (*bpppb*)

3. Estándar CCSDS-123

El *Consultative Committee for Space Space Data System* es un consorcio de las principales agencias espaciales del mundo que ha redactado tres estándares o recomendaciones para la compresión de datos en el espacio: una solución de compresión sin pérdidas universal [12], un estándar de compresión con pérdidas o sin pérdidas para imágenes bidimensionales [13] y un algoritmo de compresión para imágenes multiespectrales e hiperespectrales sin pérdidas, CCSDS-123.

El estándar CCSDS-123.0B-1, establece por tanto el método a aplicar para la compresión de datos sin pérdidas de las imágenes de tres dimensiones obtenidas por los instrumentos utilizados en misiones espaciales, así como establece el propio formato de compresión.

Las imágenes a la entrada del compresor deben cumplir una serie de características en cuanto a su formato y organización. A la salida de la compresión se obtiene un bitstream a partir del cual es posible obtener la imagen original. El bitstream resultante depende de diversos parámetros y de la propia imagen a comprimir, por lo que, aunque se especifica su formato, su longitud es variable.

El estándar CCSDS-123.0B-1 establece que las imágenes de entrada deben ser imágenes tridimensionales de hasta $2^{16} \times 2^{16} \times 2^{16}$ muestras, presentando dichas muestras un rango dinámico de bits no inferior a 2 y no superior a 16. El estándar no contempla aspectos como el posible reordenamiento de bandas dentro de la imagen, o el posible particionado en bloques de imágenes que se compriman de forma independiente. Estos aspectos influyen en la efectividad de la compresión, y sirven para limitar el impacto de posibles pérdidas de datos o corrupción en los canales de comunicación, por lo que pueden ser considerados e incluidos en la compresión como parte adicional al estándar.

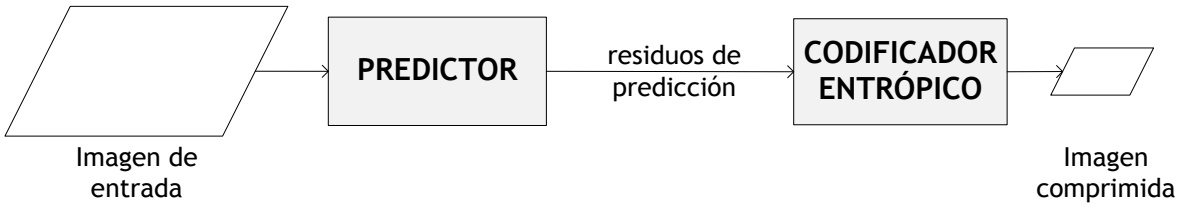

Fig. 5: CCSDS-123 - Bloques funcionales principales

La compresión está descrita como dos etapas principales, como muestra

Fig. 5:

- Predicción

  Esta etapa consiste en un método de predicción adaptativa lineal para la estimación de cada una de las muestras de la imagen a partir de muestras vecinas. Una vez calculadas esas predicciones, se mapean sus residuos, es decir, las diferencias entre los valores de las muestras reales y los estimados, a un entero sin signo del mismo tamaño que los valores de entrada. Estos residuos mapeados constituyen la salida de la etapa de predicción.

- Codificación

  En esta etapa se crea el bitstream de la imagen comprimida, que consiste en una cabecera seguida de los códigos de palabra asignados a los residuos de predicción mapeados en la etapa anterior.

3.1. Predicción

Como se ha especificado, el estándar establece una primera parte de predicción, con el cálculo de unos valores de muestras predichos $\{\hat{s}_{x,y,z}\}$ y unos residuos de predicción mapeados $\{\delta_{x,y,z}\}$, a partir de las muestras que conforman la imagen de entrada $\{s_{x,y,z}\}$. El flujo de trabajo se ilustra en Fig. 6, y se detalla a continuación.

Para cada una de las bandas de la imagen, el predictor calcula una serie de sumas locales a partir de valores de muestras vecinas, *local sum*, y que a su vez sirven para calcular unas diferencias locales, *local differences*, todo ello relativo a cada muestra de la imagen. Los valores predichos se

corresponden con una relación entre las sumas locales de la banda en cuestión y una suma ponderada de diferencias locales de la propia banda y bandas anteriores. Estos valores de ponderación, *weights*, se deben actualizar a medida que se realiza cada predicción. Una vez obtenida la predicción se mapean los residuos $\{\delta_{x,y,z}\}$, como se indicó anteriormente.

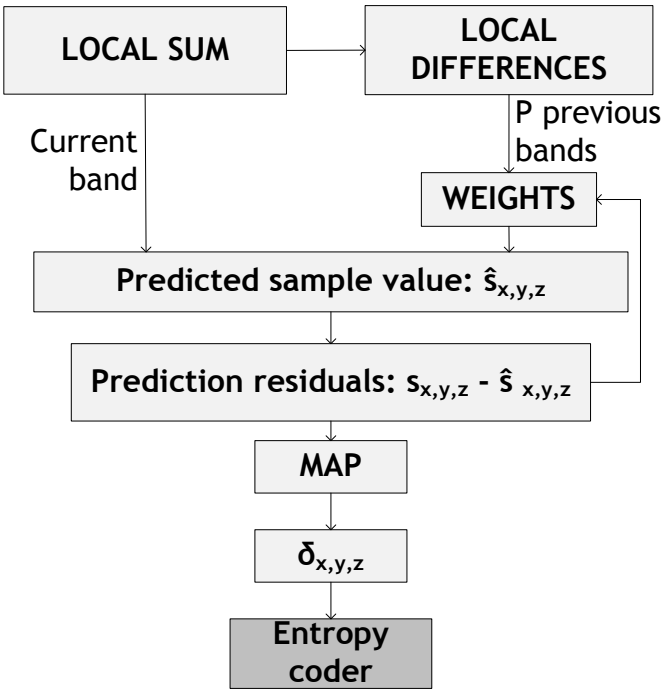

Fig. 6: CCSDS-123 - Flujo de trabajo

La suma local es una suma ponderada de muestras vecinas adyacentes a la muestra a calcular, cuyo cómputo se contempla en el estándar a través de dos posibles procedimientos, como se observa en Fig. 7.

- Orientada por vecinos: donde la suma local se corresponde con la suma de los valores de muestras previas adyacentes de la misma banda, salvo excepciones por geometría.
- Orientada por columna: donde la suma local es directamente cuatro veces el valor de la muestra vecina superior (misma columna de la línea anterior).

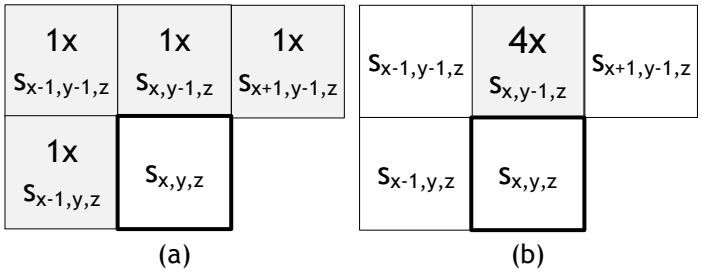

Fig. 7: CCSDS-123 - Orientación para el cálculo de Sumas Locales

(a) Orientado por vecinos (b) Orientado por columnas

Para el cómputo de las diferencias locales, se tiene en cuenta su origen, dando lugar a dos posibles tipos de diferencias:

- Diferencia local central: para cada banda, es la diferencia entre la suma local calculada para cada muestra y cuatro veces el valor real de la muestra.
- Diferencias locales direccionales: se corresponden con la diferencia entre la suma local calculada para una muestra y cuatro veces el valor de una de las muestras vecinas denominadas *N, W,* o *NW,* como se aprecia en Fig. 8.

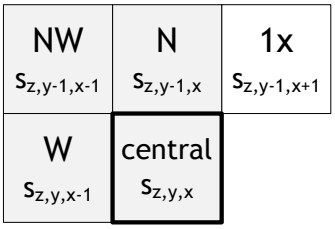

Fig. 8: CCSDS-123 - Diferencias locales en una banda espectral

De acuerdo al estándar, se establece como parámetro *P* el número de bandas previas a usar en la etapa de predicción para el cálculo de las diferencias locales; es el usuario quien determina la cantidad de bandas, dentro de un máximo permitido de 15. También se permite caracterizar los factores de ponderación aplicados en esta etapa.

El estándar no especifica un tipo de predicción en concreto, si no las posibles configuraciones para realizar dicha predicción. Así, por ejemplo, el modo *full* o el modo *reduced* hacen referencia al uso exclusivo de diferencias

locales centrales y a una combinación de diferencias centrales locales y diferencias locales direccionales, respectivamente; cada modo y en función de la imagen de entrada y del resto de configuraciones de predicción, permite obtener mejores o peores resultados de compresión, pero es labor de los usuarios establecer un formato de predicción en concreto de entre los posibles.

De acuerdo a la configuración elegida, se calcula para cada muestra un vector de diferencias locales que contendrá la parte correspondiente a las sumas locales y diferencias locales elegidas y relativas a tantas bandas previas como se haya especificado. A modo de resumen, Fig. 9 muestra cómo se calculan los vectores de diferencias locales para las posibles configuraciones, que servirán para calcular la predicción de las distintas muestras.

				Vector de diferencias locales para banda z	
				Orientado a vecinos	Orientado a columnas
Predicción en modo *reduced*	Predicción en modo *full*	Diferencias locales centrales		$(4s_{x,y} - (s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1}))_{z-1}$	$(4s_{x,y} - 4s_{x,y-1})_{z-1}$
				$(4s_{x,y} - (s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1}))_{z-2}$	$(4s_{x,y} - 4s_{x,y-1})_{z-2}$
				. . .	. . .
				$(4s_{x,y} - (s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1}))_{z-P}$	$(4s_{x,y} - 4s_{x,y-1})_{z-P}$
		Diferencias locales direccionales	N	$(4s_{x,y-1} - (s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1}))_z$	$(4s_{x,y-1} - 4s_{x,y-1})_z$
			W	$(4s_{x-1,y} - (s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1}))_z$	$(4s_{x-1,y} - 4s_{x,y-1})_z$
			NW	$(4s_{x-1,y-1} - (s_{x-1,y} + s_{x-1,y-1} + s_{x,y-1} + s_{x+1,y-1}))_z$	$(4s_{x,y-1} - 4s_{x,y-1})_z$

Fig. 9: CCSDS-123 - Configuraciones del predictor

3.2. Codificación

El estándar especifica tanto la etapa de codificación como el formato de la imagen comprimida resultante. La estructura de imagen comprimida está compuesta por una cabecera seguida de un cuerpo, como se observa en Fig. 10.

Fig. 10: CCSDS-123 - Estructura de imagen comprimida

La cabecera generada de acuerdo al estándar es variable, y depende de los parámetros de compresión así como de la propia imagen, tal y como se muestra en Fig. 11; estos datos se han de enviar junto con los datos comprimidos de la imagen para su posterior descompresión.

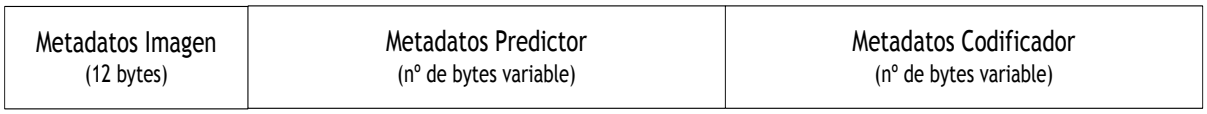

Fig. 11: CCSDS-123 - Estructura de cabecera

En cuanto al cuerpo, que está formado por los residuos de predicción mapeados y codificados, de acuerdo al estándar pueden estar dispuestos en diferentes órdenes, independientemente del orden de captura de la información o del orden de tratamiento en la etapa de predicción.

El estándar define que el proceso de codificación de los residuos puede llevarse a cabo mediante una de las siguientes opciones:

- Codificador entrópico *sample-adaptive*

Según este codificador, cada residuo se codifica usando una palabra de código binaria de longitud variable, que se calcula según valores estadísticos que se actualizan tras cada codificación y con diferentes estadísticas para cada banda. Las estadísticas consisten en un acumulador y un contador, cuya relación proporciona una estimación del valor medio de residuo mapeado, que determina a su vez la longitud variable del código de codificación. Al realizar la codificación mediante este método, el tamaño de imagen comprimida resultante no depende del orden de codificación de los residuos

- Codificador *block-adaptive*

El método de codificación se aplica, e incluso se determina, para cada conjunto de residuos denominado bloque, de tamaño especificado por el usuario, y no para cada residuo calculado en la etapa de predicción, como ocurre con la codificación de tipo *sample-adaptive*. En el caso del *block-*

adaptive, al agrupar residuos en bloques para su codificación, sí que influye el orden en el que traten.

Los residuos se pueden codificar en los siguientes órdenes:

- BI, del inglés *Band-Interleaved*

  Cada línea de la imagen se particiona a lo largo del eje *z*, formando uno o varios conjuntos de muestras, en función del parámetro de profundidad de intervalo, *M*. Los casos extremos considerados por el estándar para este parámetro, M=1 y $M=N_z$, dan lugar a los órdenes *BIP* (*Band-Interleaved-by-Pixel*, para profundidad de intervalo 1) y *BIL (Band-Interleaved-by-Line*, para profundidad de intervalo igual al número de bandas de la imagen).

- BSQ, del inglés B*and-SeQuential*

  En este caso el orden de tratamiento es el recorrido por todas las líneas de cada una de las bandas que componen la imagen.

4. Modelado del estándar en Intel CoFluent Studio

En este punto del documento se describe en líneas generales la herramienta elegida para el modelado del compresor, con el fin de introducir los conceptos básicos y distintos elementos que se detallarán en la siguiente sección, donde se explica el modelo desarrollado.

4.1. Introducción a Intel CoFluent Studio

Intel CoFluent Studio es una herramienta desarrollada por Intel capaz de modelar y simular el comportamiento, temporalidad, arquitectura, y estimaciones de prestaciones de cualquier sistema electrónico. Partiendo de unas especificaciones de entrada en código ANSI C o C++, la herramienta genera automáticamente un código SystemC, descrito a nivel de transacciones, que puede ser utilizado para verificación en cualquier testbench.

El entorno de modelado y simulación de Intel CoFluent, está compuesto principalmente por:

- Modelador gráfico que permite la captura del comportamiento del código y de la propia plataforma de ejecución.
- Entorno de simulación que permite generar e instrumentar el modelo SystemC del diseño completo, así como interpretar las trazas de simulación obtenidas.

El uso de esta herramienta permite mejorar el ciclo de vida del desarrollo de sistemas, facilitando el proceso de elección de arquitectura al independizar el modelo funcional de la plataforma, y la definición de distintos escenarios para la validación del comportamiento en tiempo real, prediciendo rendimiento y generando casos de test para la implementación.

En Fig. 12 se observa el flujo de diseño establecido por la herramienta, denominado flujo de diseño de *Y*. Los tres pasos establecidos son: creación del modelo funcional- temporal, creación del modelo de la plataforma, y un posterior modelado de la arquitectura final, donde se establecen determinadas relaciones entre el modelo funcional y el modelo de

plataforma. Tras este último paso, es posible aplicar uno de los procesos clave de Intel CoFluent Studio, que es el de la exploración. La herramienta permite realizar de forma automática una cantidad variable de ejecuciones, cada una de ellas con distinto valor para uno o varios de los parámetros de diseño, obteniendo como salida la evolución de cualquier prestación indicada por el usuario en función de la variación de dichos parámetros.

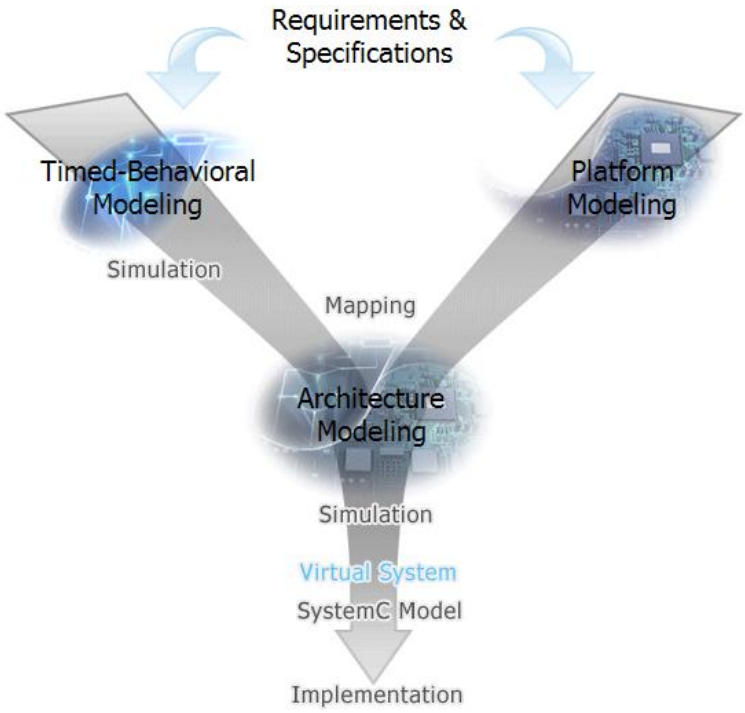

Fig. 12: Intel CoFluent Studio - Modelo de desarrollo en Y

4.1.1. Modelo de la aplicación

Este modelado, denominado *TBM* (de sus siglas en inglés en inglés, *timed-behavioral modeling)*, trata de encontrar la arquitectura adecuada desde el punto de vista de la propia aplicación. La descripción creada consiste en una estructura funcional, en la que el comportamiento de cada función es independiente de la tecnología.

El diseño se logra a través de los siguientes pasos:

- Captura de la descripción funcional a través del editor gráfico.
- Caracterización de los componentes de la solución con los atributos oportunos.

- Incorporación del comportamiento de las funciones a través de los algoritmos necesarios.
- Incorporación de parámetros de diseño.

El modelo de aplicación es en su mayoría gráfico y se basa en dos aspectos:

- La organización o estructura funcional, obtenida mediante una jerarquía de funciones o componentes y las relaciones entre ellos. Son posibles tres tipos de relaciones, tal y como aparece en la Fig. 13: variables compartidas (intercambio de información sin dependencia temporal), eventos (especificación de dependencias temporales) y transferencias de datos mediante cola de mensajes (que implican una relación de tipo productor/consumidor).

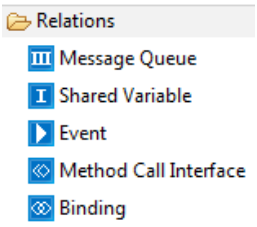

Fig. 13: Intel CoFluent Studio - Relaciones entre funciones

- El comportamiento, relativo a cada función o componente activo se especifica mediante un conjunto de operaciones y su ordenación temporal, a través de un conjunto de operadores. Los principales operadores se muestran en Fig. 14, y de entre ellos cabe destacar: *initial loop*, punto de comienzo del comportamiento especificado; *operation*, componente en el que se añade la parte algorítmica correspondiente; *loop*, incorpora la iteratividad al comportamiento; *input action* y *output action*, permiten recoger datos externos a la función o generarlos; y por último *alternative*, que permite ejecuciones condicionales.

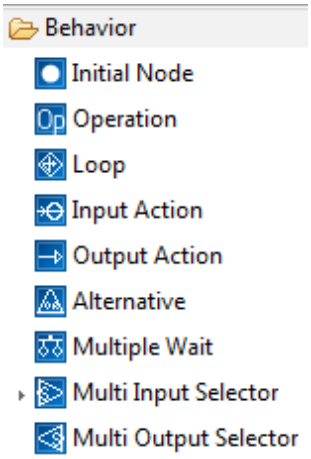

Fig. 14: Intel CoFluent Studio - Componentes de comportamiento

El modelo de ejecución, como ya se ha introducido, debe ser caracterizado mediante los atributos correspondientes de los distintos elementos que lo conforman. Estos atributos pueden ser por ejemplo el tiempo de ejecución de una operación, los tiempos de lectura y escritura de una variable compartida, capacidad de la cola de mensajes, o el periodo de activación de una determinada función.

Como paso adicional se pueden incorporar a dicho modelo los ya mencionados parámetros de diseño. Estos parámetros son variables que pueden tomar distintos valores en un rango especificado, y que se usan para el estudio de su influencia sobre una determinada métrica.

Una vez se completa la descripción del modelo, Intel CoFluent Studio genera un código en SystemC que se puede simular, ejecutar y monitorizar, permitiendo validar la descripción realizada, su estructura funcional, el modelo de comportamiento de cada función y la algorítmica asociada. Se puede también verificar la solución mediante la generación de los resultados oportunos, así como estudiar las propiedades del comportamiento de la solución y modelar los componentes de la manera más oportuna.

4.1.2. Modelo de la plataforma

Con este modelado se pretende definir la arquitectura física, o una abstracción de la plataforma hardware adecuada sobre la cual ejecutar posteriormente el modelo de aplicación. Como punto de partida, se puede

tomar tanto el modelo de aplicación como una serie de restricciones tecnológicas. Los pasos para conseguir un modelo de plataforma son los siguientes:

- Captura del sistema a través del editor gráfico.
- Caracterización de los componentes mediante la especificación de sus atributos.
- Incorporación de los parámetros de diseño.
- Creación de las posibles rutas de comunicación.

Los distintos componentes que pueden encontrarse en un sistema son los mostrados en Fig. 15.

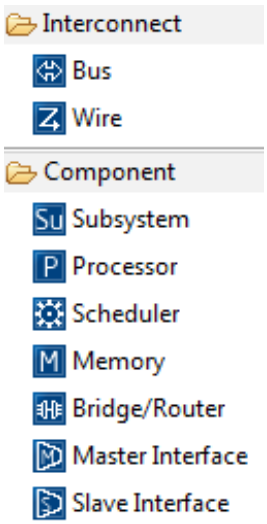

Fig. 15: Intel CoFluent Studio - Componentes del modelo de plataforma

Como elementos de interconexión se cuenta con buses y conexiones simples. Como componentes propiamente dichos cabe destacar: *procesador*, recurso físico para la ejecución de funciones; *planificador*, encargado de la programación de la ejecución de funciones; *memorias*, para el almacenamiento de información referente a variables compartidas o colas de mensajes; y por último, *interfaces maestras* o *interfaces esclavas* de comunicación, las cuales permiten iniciar una comunicación o poder recibir una comunicación iniciada por otro componente del sistema.

Los componentes del modelo de plataforma presentan, al igual que aquellos del modelo de aplicación, una serie de atributos configurables por el

usuario y que permiten caracterizar la plataforma de manera bastante precisa. Así mismo, en este nivel también se pueden incorporar parámetros de diseño con el fin de parametrizar la plataforma diseñada y facilitar el estudio de distintas configuraciones.

4.1.3. Modelo de la arquitectura

En este punto es en el que se lleva a cabo el proceso de obtención de la definición de la solución final. Consiste en el desarrollo del soporte de ejecución y su organización. El proceso, denominado mapeado, establece cómo se implementa el modelo de aplicación sobre el modelo de plataforma, y puede suponer un estudio de rendimiento, restricciones u otros factores, hasta obtener la solución deseada.

El mapeado incluye la parte de asignación referente tanto a las funciones del modelo de aplicación, como a las rutas de datos presentes en él.

Se ofrecen cuatro posibilidades para mapear funciones:

- *Not allocated*: mapeado por defecto de las funciones, en las que no se asignan a ningún procesador pero se ejecutan sobre el modelo de arquitectura tal y como describe el modelo de comportamiento.
- *Excluded*: posibilidad que supone suspender la ejecución de una función en el modelo de arquitectura.
- *Processor*: opción que ejecuta las interfaces y funciones que son mapeadas directamente sobre él; puede contener un planificador para modelar ejecución software.
- *Scheduler*: que establece la ejecución secuencial de determinadas funciones a través de una política de arbitraje establecida por el usuario.

Las rutas de datos se establecen entre cada variable compartida del modelo de aplicación y cada una de las funciones a las que está conectada, y éstas se podrán o no mapear a las distintas rutas de comunicación del modelo de plataforma.

4.1.4. Simulación y verificación

Intel CoFluent Studio permite el análisis y estudio de los diferentes elementos del diseño, generando de manera sencilla y prácticamente automática distintas representaciones entre las que se destacan:

- Timeline: diagrama que presenta el estado de todos los componentes del diseño a nivel de transacciones durante la simulación ejecutada.
- Gráficas de variables: en evolución temporal o en función de otras variables del diseño.
- Representación de variables en función de parámetros del diseño.
- Exploración: resultados comparativos sobre la variación de uno de los parámetros de diseño.

4.2. Código de referencia del estándar CCSDS-123

Como el objetivo de este Trabajo Fin de Master es modelar el compresor de imágenes multiespectrales e hiperespectrales del estándar CCSDS-123, es necesario partir de un código de referencia, que en este caso es el algoritmo de compresión que implementa compresión sin pérdidas para imágenes multiespectrales e hiperespectrales de acuerdo al estándar recomendado CCSDS-123. Este código de referencia ha sido desarrollado en la ESA y se distribuye bajo Licencia Pública de la ESA.

De acuerdo al estándar, el algoritmo se presenta dividido en dos funciones principales: una primera función de predicción y una posterior de codificación. El algoritmo recibe las características de la imagen de entrada a comprimir a través de una serie de parámetros que se detallan en Tabla 1. Así mismo, recibe los parámetros escogidos por el usuario para determinar las características de la compresión y la organización de los datos de salida.

PARÁMETRO	SIGNIFICADO	POSIBLES VALORES
input	Imagen de entrada	*imagen*
output	Imagen de salida	*imagen*
rows	Número de líneas de imagen de entrada	$[1,2^{16}]$
columns	Número de columnas de imagen de entrada	$[1,2^{16}]$
bands	Número de bandas de imagen de entrada	$[1,2^{16}]$
in_format	Formato de organización de muestras de imagen de entrada	$\{BI, BSQ\}$
in_depth	Para BI: profundidad de intervalo de entrada	$[1, bands]$
in_byte_ordering	Organización de bytes en palabras de imagen de entrada	$\{little\ \ endian, big\ \ endian\}$
out_format	Formato de organización de muestras de imagen de salida	$\{BI, BSQ\}$
out_depth	Para BI: profundidad de intervalo de salida	$[1, bands]$
dyn_range	Margen dinámico de las muestras de entrada	$[2,16]$
word_len	Tamaño en bytes de palabra de salida	$[1,8]$
reg_input	Especificación de codificación en 16 de muestras de entrada, pese a margen dinámico indicado menor	$\{reg\_input, -\}$

Tabla 1: CCSDS-123 – Parámetros generales de imagen

En cuanto a la etapa de predicción, el software permite su configuración tal y como se especifica en la recomendación del estándar. Por ejemplo, si se desea usar sumas locales orientadas por vecinos o por columnas, se debe indicar mediante el parámetro *neighbour_sum*; si se desean usar únicamente diferencias locales centrales para la predicción, debe omitirse el parámetro *full*.

A continuación, en Tabla 2 se detallan los parámetros de la etapa de predicción.

PARÁMETRO	SIGNIFICADO	POSIBLES VALORES
signed_sample	Especificación de muestras de entrada como enteros positivos	$\{signed\_samples, -\}$
pred_bands	Número de bandas usadas para la predicción	$[1,15]$
full	Si se especifica, se usará modo de predicción *full*	$\{full, -\}$
neigbour_sum	Especificación de sumas locales calculadas a partir de valores de muestras vecinas	$\{neighbour\_sum, -\}$
reg_size	Tamaño de registro	$[1,64]$
w_resolution	Resolución del valor de ponderación	$[4,19]$
w_interval	Valor del incremento de actualización del factor de ponderación	$[4,11]$
w_initial	Valor inicial del factor de ponderación	$[-6,9]$
w_final	Valor final del factor de ponderación	$[-6,9]$
w_init_resolution	Resolución del factor de ponderación inicial	$[3, weight\_resolution + 3]$

Tabla 2: CCSDS-123 – Parámetros de predicción

Se desea remarcar en este punto que la predicción se realiza en el estándar como un único paso de cara al usuario, es decir, se calculan los valores mapeados de todas las muestras de la imagen, y es entonces cuando se pasa la matriz de dichos valores a la etapa de codificación, que está formada por el mismo número de elementos que la imagen original.

Por último, los parámetros necesarios para la etapa de codificación se detallan en Tabla 3, y los específicos del codificador *Block Adaptive*, en Tabla 4.

PARÁMETROS	SIGNIFICADO	POSIBLES VALORES
sample_adaptive	Especificación de codificador *Sample Adaptive*	$\{sample\_adaptive,-\}$
u_max	Para *Sample Adaptive*: límite de longitud unaria	$[8,32]$
y_star	Para *Sample Adaptive*: intervalo de re-escalado de contador y acumulador	$[máx(4, y\_0),9]$
y_0	Para *Sample Adaptive*: exponente de contador inicial	$[1,8]$
k	Para *Sample Adaptive*: constante de inicialización de acumulador	$[0, dyn\_range-2]$
k_init_file	Para *Sample Adaptive*: tabla de inicialización de acumulador	$\{tabla,-\}$

Tabla 3: CCSDS-123 - Parámetros del codificador *Sample Adaptive*

PARÁMETROS	SIGNIFICADO	POSIBLES VALORES
block_size	Tamaño de bloque de codificación	$\{8,16,32,64\}$
restricted_enc	Modo restringido	$\{restricted\_enc,-\}$
ref_interval	Intervalo de referencia	$(0,4096]$

Tabla 4: CCSDS-123 - Parámetros del codificador *Block Adaptive*

4.3. Modelo de aplicación

El diseño en CoFluent se ha realizado en base a las funciones presentes del código de referencia y manteniendo su jerarquía. A continuación se describe la solución obtenida, los componentes presentes y su comportamiento.

El diseño del compresor se estructura en tres funciones, tal y como se puede observar en Fig. 16. Las comunicaciones entre las distintas funciones se realizan mediante variables compartidas con los correspondientes datos de información y de control, así como a través de distintos eventos, que permiten realizar operaciones en paralelo, en la medida de lo posible, de las distintas imágenes que lleguen al sistema.

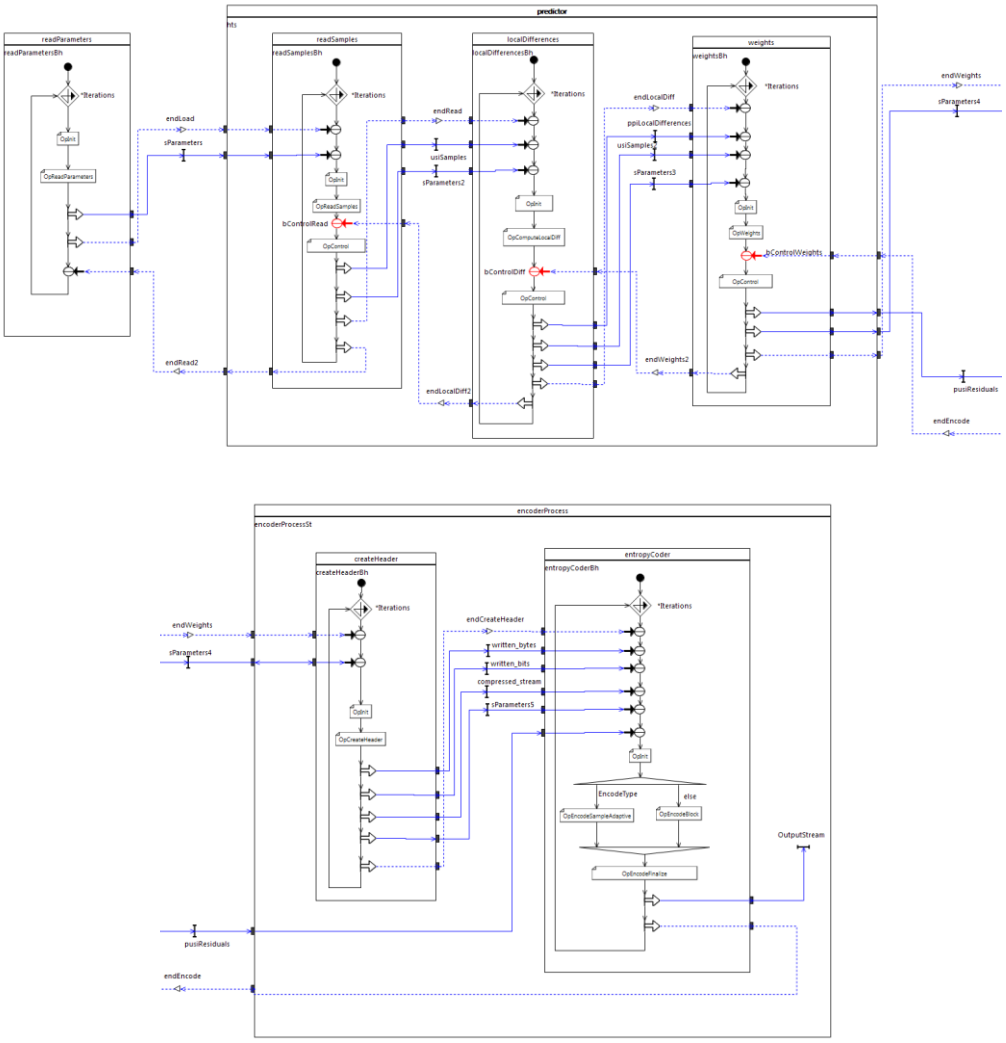

Fig. 16: CCSDS-123 CoFluent - Modelo de aplicación

- *readParameters*

  Es la función encargada de obtener los parámetros que describen la imagen de entrada que se desea comprimir, los parámetros de configuración tanto para el proceso de predicción y como para el proceso de codificación. Con el fin de obtener un modelo lo más fidedigno posible al código de referencia, y que permita el mismo nivel de configuración, a la vez que facilite el análisis de cualquier simulación que se desee realizar para su verificación, se han incluido como parámetros de diseño, aquellos que aparecen en el código de referencia y son de trascendencia para las simulaciones, y que se muestran en Tabla 5.

- *predictor*

Función encargada de ejecutar la fase de predicción de las muestras que conforman la imagen hiperespectral. Se divide en tres sub-funciones que son: *readSamples*, *localDifferences* y *weights*. En cada una de estas tres funciones se realizan pasos necesarios para la compresión, y por tal y como se refleja en en el código de referencia, las dependencias de datos existentes entre estas funciones obligan a que éstas trabajen de manera secuencial.

- *encoderProcess*

  Última función encargada de la compresión de los residuos calculados por las funciones descritas anteriormente. Compuesta por *createHeader*, sub-función encargada de generar la cabecera que precede a los datos comprimidos que conforman la imagen, y *encodingProcess*, sub-función que realiza la codificación entrópica.

A continuación se enumeran y detallan las distintas variables del modelo, Tabla 5, así como los parámetros de diseño definidos en el sistema, Tabla 6.

VARIABLES	SIFNIFICADO	TIPO
DefsParam	Estructura en la que se almacenan los parámetros de configuración de entrada, predicción y codificación	char * cSamplesFile; input_feature_t input_params; encoder_config_t encoder_params; predictor_config_t predictor_params; char * cOutFile; DefData sTimes; DefDataSize DataSize;
DefusiSamples	Imagen a comprimir	unsigned short int*
DefppiLocalDifferences	Matriz n-dimensional que almacena las diferencias locales calculadas	int**
DefuiWrittenBits	Número de bits del bitstream	unsigned int

VARIABLES	SIFNIFICADO	TIPO
DefuiWrittenBytes	Número de bytes del bitstream	unsigned int
ucpOutputStream	Bitstream que contiene la cabecera y la imagen comprimida	unsigned char *

Tabla 5: CCSDS-123 CoFluent - Variables del modelo

PARÁMETRO	TIPO	VALORES	VALORES POR DEFECTO
Iterations	Variable	$[1,20]$	20
PreBands	Rango	$[1,15]$	3
ImageFile	Rango	$\{set\ de\ imágenes\}$	
XSize	Variable	$[1,2^{16}]$	16
YSize	Variable	$[1,2^{16}]$	16
ZSize	Variable	$[1,2^{16}]$	21
EncodeType	Set	$\{0:block,1:sample\_adaptive\}$	0

Tabla 6: CCSDS-123 CoFluent – Parámetros de diseño del modelo

A modo aclaratorio es necesario explicar que la temporalidad de los distintos procesos asociados a cada algoritmo, se ha estimado a través de funciones propias del sistema operativo Windows creadas para tal fin, y en el entorno de Microsoft Visual Studio sobre el código de referencia. Las funciones se encuentran en el archivo *windows.h* y se han utilizado en los puntos concretos de inicio y finalización de operación para cada sección de código a temporalizar. Estos tiempos estimados influyen directamente en las prestaciones observadas del modelo, por lo que se han realizado medidas de tiempo de manera repetitiva y finalmente se ha aplicado una media.

4.3.1. readParameters

Esta función, que se ilustra en Fig. 17, y que se encarga de extraer los parámetros de imagen, predicción y codificación, comienza con una serie de inicializaciones, en la operación *OpInit*, y a continuación realiza una rutina

sobre los parámetros de diseño, para almacenar la información oportuna en las correspondientes variables, que se encuentran recogidos en una variable de tipo *DefsParam, sParameters*, y avisando del fin de operación mediante el evento *endLoad*.

Este proceso se repite el número de veces indicado por *Iterations,* definido como parámetro de diseño para simular un número determinado de ejecuciones del algoritmo completo, como si existiese implícita una partición de la imagen de entrada, y con una frecuencia de repetición controlada por el evento *endRead2*, que avisa de la disponibilidad de la siguiente función para recibir nuevos parámetros para una nueva compresión.

En Tabla 7 se detallan los componentes de esta función y sus características.

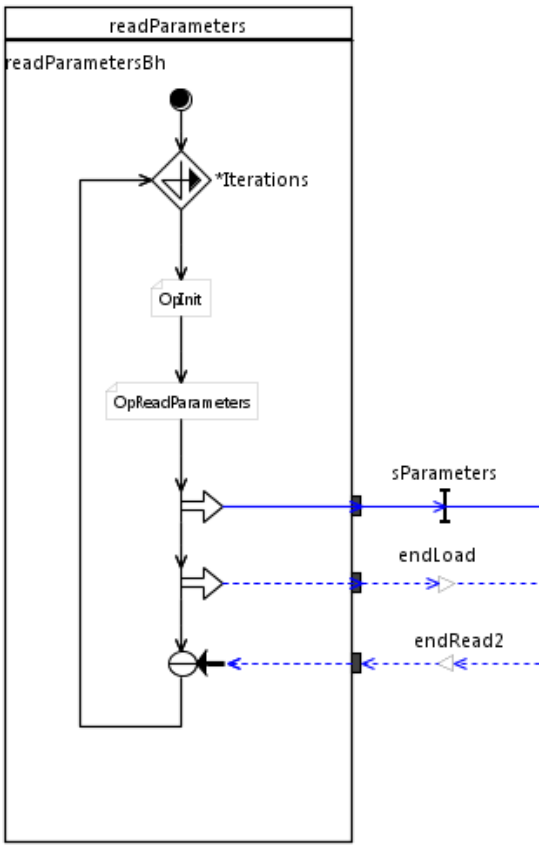

Fig. 17: CCSDS-123 CoFluent – *readParameters*

COMPONENTE: Bucle	TIPO
Bucle	(*for*)
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN**
OpInit	20 ns
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN**
OpReadParameters	20 ns

Tabla 7: CCSDS-123 CoFluent - Componentes predictor-*readParameters*

4.3.2. predictor

A continuación, se detallan las distintas sub-funciones que componen la etapa de predicción del modelo desarrollado.

4.3.2.1. readSamples

La función *readSamples*, comienza tras la señalización del evento *endLoad*, y se muestra en Fig. 18. Con este evento el sistema recoge los parámetros obtenidos en la etapa anterior de lectura de parámetros, *sParameters*, y comienza su ejecución. Tras las inicializaciones correspondientes, *OpInit*, se realiza la lectura de las muestras de la imagen de entrada mediante la operación *OpReadSamples*; en este caso, la implementación de la funcionalidad se lleva a cabo mediante una fuente externa del proyecto y a través de su llamada desde el algoritmo de la operación. Esta fuente externa se encarga del manejo del fichero de entrada que contiene los datos de la imagen que se va a comprimir.

A continuación se verifica si ha habido lectura de muestras con anterioridad, de manera que esta etapa sólo propaga resultados en caso de que se trate de la primera compresión, o bien en sucesivas compresiones si la siguiente etapa está preparada para ello, mediante la comprobación del evento *endLocalDiff2*. La propagación de muestras de imagen se realiza mediante la variable compartidas *usiSamples* y los parámetros correspondientes mediante *sParameters2*. Se envía la señalización a la función anterior *readParameters* y a la función posterior *localDifferences* mediante los eventos *endRead2* y *endRead*, respectivamente. Este conjunto de operaciones se repite igualmente las veces indicadas por el parámetro de

diseño *Iterations*.

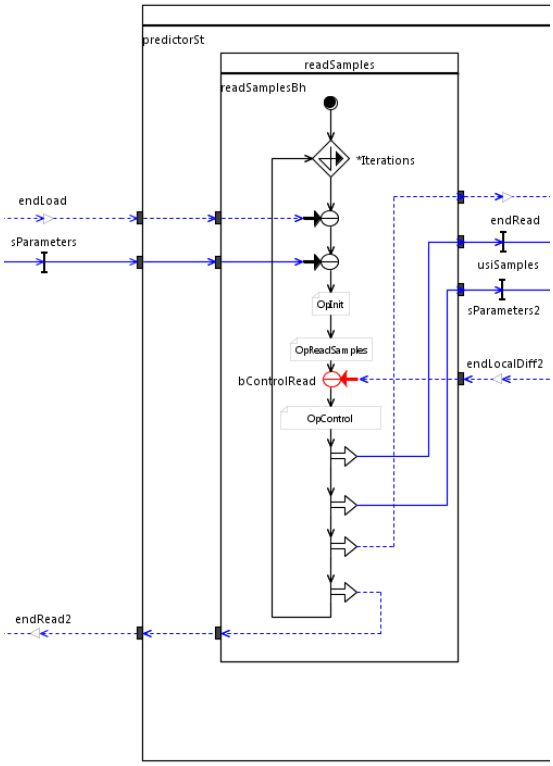

Fig. 18: CCSDS-123 CoFluent - *predictor-readSamples*

Tabla 8 recoge los componentes de esta sub-función.

COMPONENTE: Bucle	TIPO
Bucle	(*for*)
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN (ns)**
OpInit	20
OpReadSamples	*fReadSamplesTime*
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN (ns)**
OpControl	1

Tabla 8: CCSDS-123 CoFluent – Componentes *predictor-readSamples*

fReadSamplesTime se ha definido cono una variable cuyo valor se calcula para cada lectura de muestras como se indica a continuación:

$$fReadSamplesTime = 60 \times \text{input\_params.x\_size} \times \text{input\_params.y\_size} \times \text{input\_params.z\_size}$$

(Ecuación 4)

4.3.2.2. localDifferences

La función *localDifferences* realiza el cálculo de diferencias locales para las muestras de la imagen a comprimir. Por tanto, comienza su tarea una vez indicada la finalización de la etapa anterior mediante el evento *endRead*, y con la posterior lectura de las muestras a través de la variable *usiSamples* y de los parámetros de compresión mediante *sParameters2*.

Tras las inicializaciones de *OpInit*, se calculan las diferencias locales mediante el algoritmo descrito en *OpComputeLocalDiff*, y a continuación, se comprueba si ha tenido lugar el evento de finalización de la siguiente etapa, para en caso positivo propagar las variables correspondientes: diferencias locales calculadas a través de *ppiLocalDifferences*, muestras de la imagen mediante *usiSamples2*, los parámetros del proceso mediante *sParameters3*.

La finalización de cada cálculo de diferencias locales para cada imagen, se señaliza a las etapas *readSamples* y *weights* mediante *endLocalDiff2* y *endLocalDiff*, respectivamente.

En Fig. 19 se aprecian los distintos procesos descritos así como las variables y eventos de esta etapa del modelo del compresor; los componentes se recogen en Tabla 9.

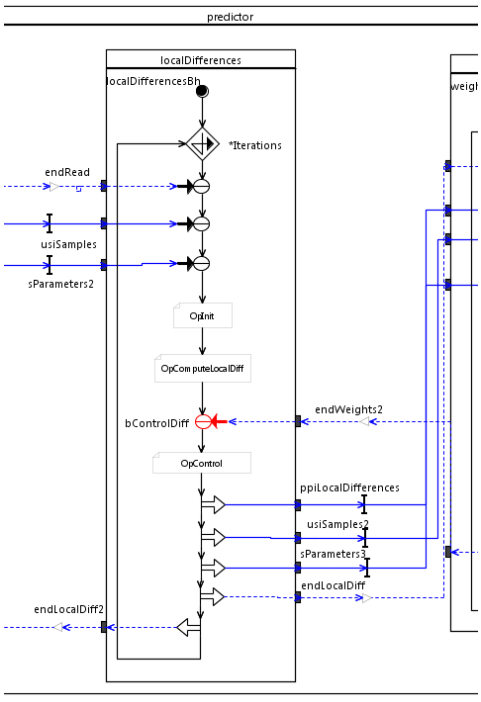

Fig. 19: CCSDS-123 CoFluent - *predictor-localDifferences*

COMPONENTE: Bucle	TIPO
Bucle	(*for*)
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN (ns)**
OpInit	10
OpComputeLocalDifferences	*fLocalDifferencesTime*
OpControl	1

Tabla 9: CCSDS-123 CoFluent - Componentes *predictor-localDifferences*

$$fLocalDifferencesTime = 53.04 \times \text{input\_params.x\_size} \times \text{input\_params.y\_size} \times \text{input\_params.z\_size} \quad \text{(Ecuación 5)}$$

4.3.2.3. weights

La función *weights*, se encarga de las tareas de cálculo de la muestra predicha y de los correspondientes residuos de predicción, su mapeado y de la actualización de los pesos, con los datos propagados y calculados por etapas anteriores. En Fig. 20 se puede observar el diseño de esta función.

La función *weights* comienza su operación tras la señalización de la etapa *localDifferences*, recogiendo las variables con las diferencias locales, las muestras de la imagen y los parámetros de compresión (evento *endLocalDiff*, variables *ppiLocalDifferences*, *usiSamples2* y *sParameters3*).

OpInit, como en otras etapas, contiene las inicializaciones de la etapa. El contenido algorítmico de *OpWeights*, contiene las llamadas a la fuente externa correspondiente a los procesos anteriormente mencionados. El control final para propagación de parámetros es análogo al de etapas anteriores: depende del evento de finalización *endEncode*, que proviene de la sub-función de codificación de la etapa siguiente, y afecta a las variables *pusiResiduals* y *sParameters4*. Los eventos de señalización de salida, *endWeights* y *endWeights2* van conectados a *createHeader* y *localDifferences*, respectivamente.

En este punto del diseño se considera necesario aclarar el contexto de la variable *pusiResiduals*; éste se extiende a todos y cada uno de los residuos de

predicción de todas las muestras de la imagen a comprimir, por lo que sólo se genera y propaga desde esta etapa una única vez por imagen.

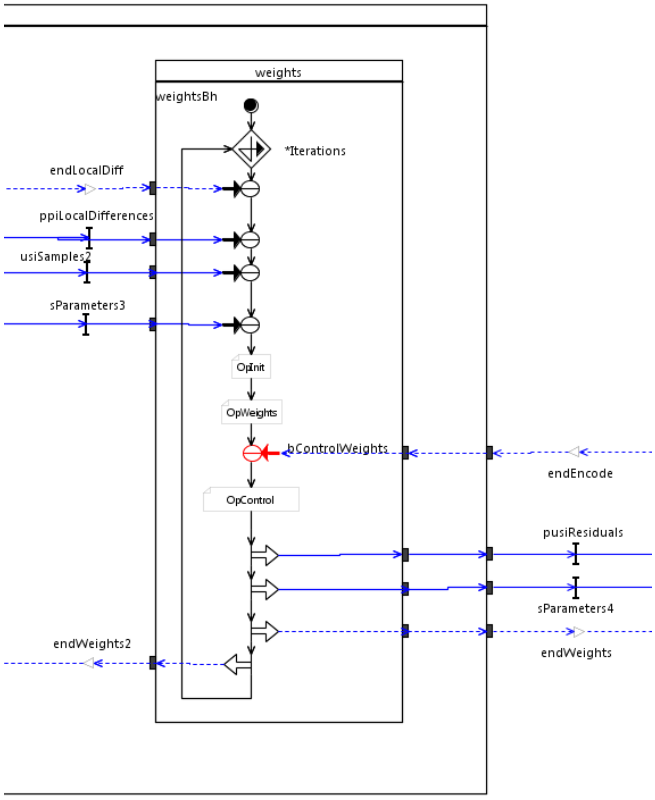

Fig. 20: CCSDS-123 CoFluent - *predictor-weights*

Los componentes asociados a la última sub-función de la función de predicción, aparecen en Tabla 10 a continuación.

COMPONENTE: Bucle	TIPO
Bucle	(*for*)
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN (ns)**
OpInit	10
OpWeights	*fWeightsTime*
OpControl	1

Tabla 10: CCSDS-123 CoFluent - Componentes *predictor-weights*

$$fWeightsTime = 28 \times \text{input\_params.x\_size} \times \text{input\_params.y\_size} \times \text{input\_params.z\_size} \times \text{predictor\_params.pred\_bands} \quad \text{(Ecuación 6)}$$

4.3.3. encoderProcess

La última función del modelo, *encoderProcess*, se compone de dos las sub-funciones *createHeader* y *entropyCoder*, que se explican a continuación.

4.3.3.1. createHeader

Con la función de creación de cabecera comienza la generación del bitstream de salida del sistema. Una vez calculados los residuos y su correspondiente mapeado, indicado mediante el evento *endWeights*, se recogen los parámetros de compresión y los residuos, a través de *sParameters4* y *pusiResiduals*. Tras las inicializaciones de *OpInit* se lleva a cabo la generación de la cabecera que precederá al cuerpo del bitstream, a través de los pasos indicados en el apartado de algoritmo de *OpCreateHeader*.

Tras ello, y sin ninguna condición, ya que la creación de la cabecera y la codificación de los residuos se realizan de manera secuencial y no en paralelo, se propagan las variables relativas al *bitstream* ya iniciado: *written_bytes*, *written_bits* y *compressed_stream*; los parámetros del proceso sobre *sParameters5* y el evento de finalización de función *endCreateHeader* hacia *encoderProcess*. Todo este flujo queda reflejado en Fig. 21.

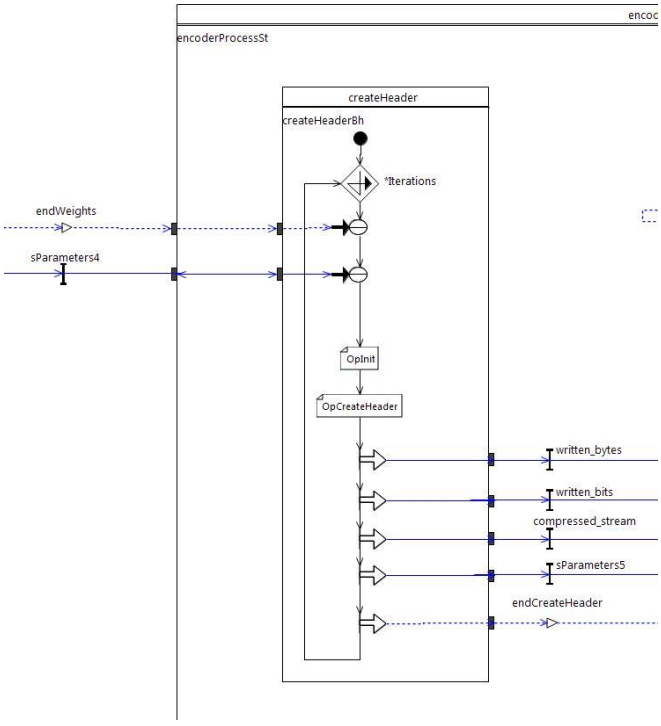

Fig. 21: CCSDS-123 CoFluent – *encoderProcess-createHeader*

Los componentes involucrados en esta etapa se especifican en Tabla 11.

COMPONENTE: Bucle	TIPO
Bucle	(*for*) {Iterations}
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN (ns)**
OpInit	10
OpCreateHeader	*fCreateHeaderTime*

Tabla 11: CCSDS-123 CoFluent - Componentes *encoderProcess-createHeader*

$$fCreateHeaderTime = 7200$$

4.3.3.2. entropyCoder

La última parte del diseño, que se corresponde con la codificación de los residuos mapeados, sigue el comportamiento del diseño que se muestra en Fig. 22.

Igual que etapas anteriores, el número de repeticiones de esta sub-función queda definido por *Iterations*, y depende de la señalización del evento *endCreateHeader* procedente de la sub-función *createHeader*. Tras esta señalización, se recogen los parámetros y variables necesarias para el proceso de codificación: *written_bits*, *written_bytes* y *compressed_stream*, para seguir rellenando el bitstream de imagen comprimida; *sParameters* para acceder a los valores de los parámetros de compresión; y finalmente *pusiResiduals* para acceder a los valores de residuos mapeados para su codificación.

Tras las inicializaciones pertinentes en *OpInit*, se realiza o bien una codificación a través del método *Sample Adaptive*, o bien una codificación a través del método *Block Adaptive*; esta posibilidad de codificación queda reflejada en el diseño a través del componente de alternativa que precede a las operaciones *OpEncodeSampleAdaptive* y *OpEncodeBlock*.

Tras la codificación de los residuos, se añade una última operación, denominada *OpEncodeFinalize*, en cuyo apartado de algoritmo se inserta el control de las variables, así como la parte correspondiente a las medidas temporales que se quieren recoger en las simulaciones.

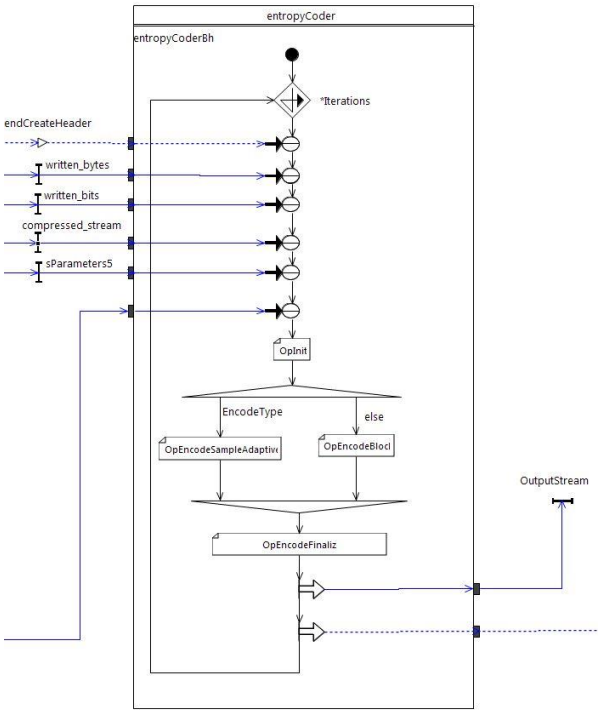

Fig. 22: CCSDS-123 CoFluent - *encoderProcess-entropyCoder*

El diseño se ha realizado para permitir las siguientes medidas:

Latencia: tiempo transcurrido desde que se registra la petición de compresión de imagen y la salida de la imagen codificada, expresado en nano-segundos, ns.

$$Latencia = t(salida\ imagen\ comprimida) - t(lectura\ primera\ muestra) \quad \text{(Ecuación 7)}$$

Throughput: permite establecer una medida de Mega-muestras por segundo para el compresor, en función del tiempo transcurrido entre salidas sucesivas de imágenes comprimidas y la cantidad de muestras comprimidas. Este tiempo depende de la configuración de predicción y codificación elegida, así como de la imagen de entrada, y se expresa en Mega-muestras por segundo, MSps.

$$Throughput = \frac{nº\ de\ muestras}{t(salida\ imagen\ n+1) - t(salida\ imagen\ n)} \quad \text{(Ecuación 8)}$$

Fig. 23 refleja gráficamente el cálculo de estos tiempos para compresiones consecutivas.

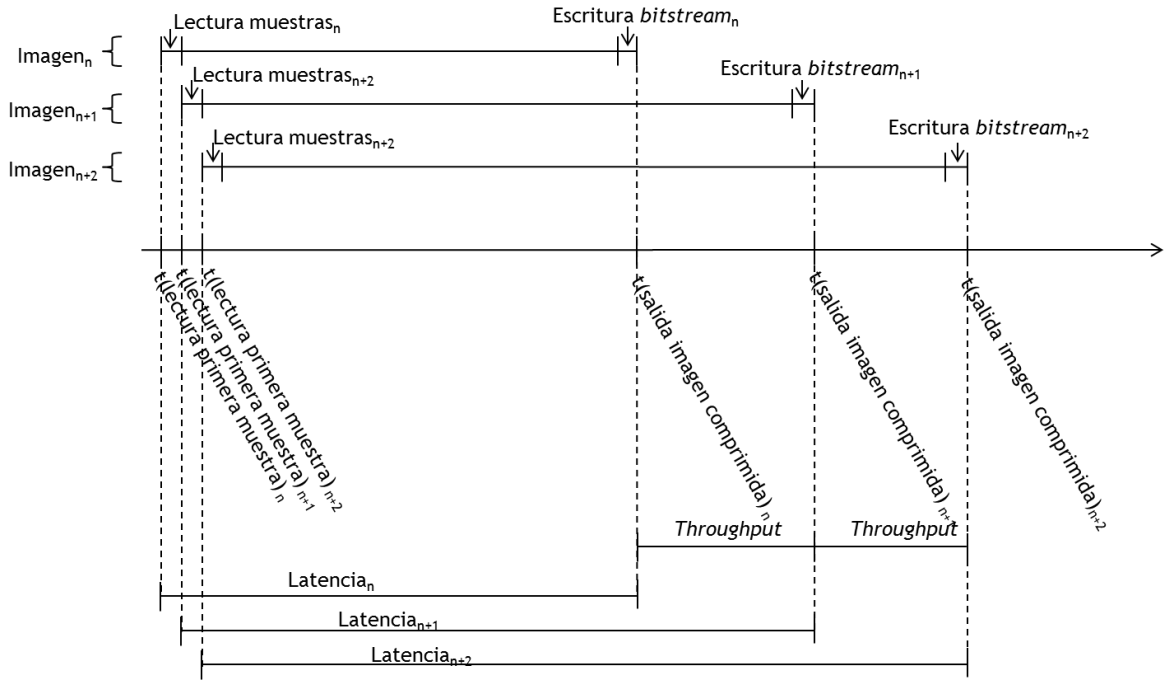

Fig. 23: CCSDS-123 CoFluent - Cálculo de tiempos

Los distintos componentes de *entropyCoder* se muestran en Tabla 12.

COMPONENTE: Bucle	**TIPO**
Bucle	(*for*) {Iterations}
COMPONENTE: Alternativa	**CONDICIÓN**
Alternativa	EncodeType (*sample*||*block*)
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN (ns)**
OpInit	10
OpEncodeSampleAdaptive	*fEncoderTime*
OpEncodeBlock	*fEncoderTime*
OpEncodeFinalize	10

Tabla 12: CCSDS-123 CoFluent - Componentes *encoderProcess-entropyCoder*

$fEncoderTime$:

$$\begin{Bmatrix} BlockAdaptive: 0{,}149 \times \text{input\_params.x\_size} \times \text{input\_params.y\_size} \times \text{input\_params.z\_size} \\ SampleAdaptive: 0{,}41 \times \text{input\_params.x\_size} \times \text{input\_params.y\_size} \times \text{input\_params.z\_size} \end{Bmatrix}$$

(Ecuación 9)

4.4. Modelo de plataforma

Con el objetivo de observar el comportamiento del algoritmo ejecutado

sobre una determinada plataforma, y observar determinados parámetros que puedan influir en su comportamiento o prestaciones, se han desarrollado dos ejemplos de plataformas. Se ha optado por las siguientes, sobre las cuales analizar el código del compresor:

- una plataforma software con número de núcleos variable (*simplePlatform*).
- una segunda plataforma en la que se cuenta con dos procesadores, uno software y otro hardware con factor de aceleración también variable (*combinedSWHWPlatform*).

4.4.1. simplePlatform

La primera plataforma, consiste en un único procesador (con número de núcleos parametrizable) sobre el que se ejecutará la aplicación modelada en la etapa anterior del flujo de diseño de la herramienta. Sobre este procesador se ha incluido un componente de planificación, al existir distintas tareas que en principio podrían ejecutarse en paralelo; el diseño se observa en Fig. 24.

Al tratarse de una plataforma sencilla, formada por un solo procesador, con la memoria asociada para los datos necesarios para la ejecución de la aplicación, no contamos con elementos que permitan la parametrización de la plataforma, por lo que solamente se puede jugar con el número de cores asociados, que impide o permite la ejecución en paralelo de un número determinado de distintas tareas, en función del número de núcleos establecidos.

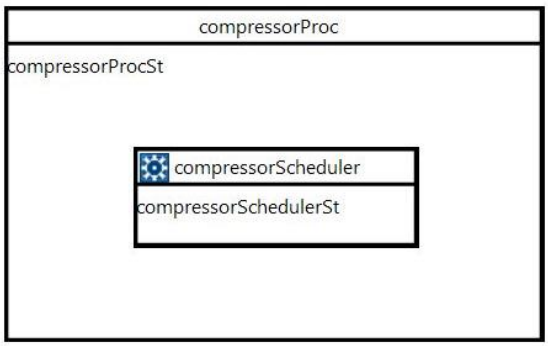

Fig. 24: CCSDS-123 CoFluent – *simplePlatform*

A continuación en Tabla 13 se detallan los dos componentes de la plataforma, y en Tabla 14, el parámetro de diseño.

COMPONENTE: Procesador	Características
compressorProc	- Factor de aceleración: 1 - Memoria disponible: 1 GB
COMPONENTE: Planificador	**Características**
compressorScheduler	- Nº de núcleos: *NumCores* - Política: Prioridad - Tiempo de planificación: 10 ns - Tiempo de slot: 10 ms

Tabla 13: CCSDS-123 CoFluent - Componentes *simplePlatform*

PARÁMETRO	SIGNIFICADO	TIPO	VALORES / VALORES POR DEFECTO	
NumCores	Número de núcleos para ejecución	Rango	[1,6]	1

Tabla 14: CCSDS-123 CoFluent - Parámetros de diseño *simplePlatform*

4.4.2. combinedSWHWPlatform

La segunda plataforma diseñada incluye un elemento más, un co-procesador hardware que permitirá acelerar el proceso de codificación.

Como se muestra en Fig. 25, al procesador software se le asigna la primera parte del proceso de compresión, que por resultados se establece como la parte más "ligera" del proceso, como se verá más adelante. Este procesador, *predictorProc*, cuenta con una interfaz para la comunicación con *Bus*, de tipo maestra, M*predictorProc*. El co-procesador hardware dedicado a la parte de codificación, que presenta un factor de aceleración, para poder simular la aceleración que supondría llevar esa parte del algoritmo a hardware, presenta a su vez una interfaz análoga a la del primer procesador, de tipo esclava y denominada *SIencoderProc*.

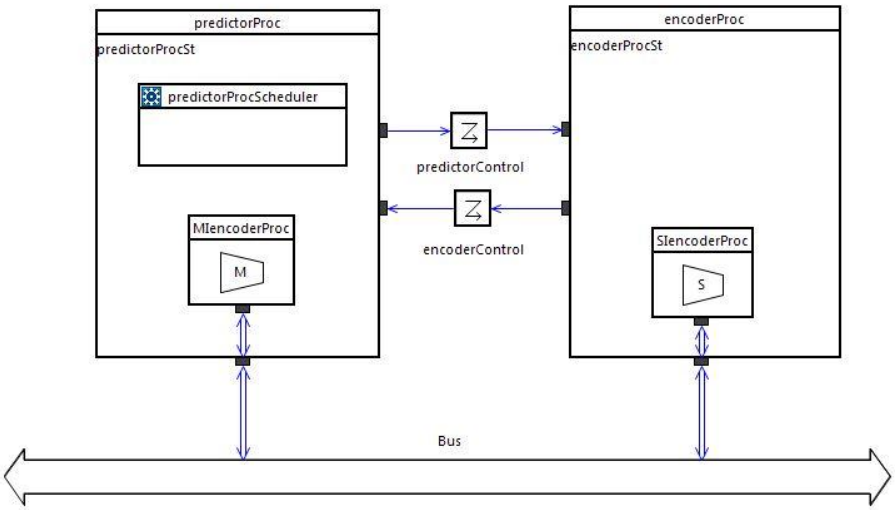

Fig. 25: CCSDS-123 CoFluent – *combinedSWHWPlatform*

COMPONENTE: procesador	Características
predictorProc	- Factor de aceleración: 1 - Memoria disponible: 1 GB
encoderProc	- Factor de aceleración: 1-6 - Memoria disponible: 1 GB
COMPONENTE: planificador	**Características**
compressorScheduler	- Nº de núcleos: *NumCores* - Política: Prioridad - Tiempo de planificación: 10 ns - Tiempo de slot: 10 ms
COMPONENTE: interfaces	**Características**
MIencoderProc	- Buffer policy: FIFO - Buffer capacity: 1
SIencoderProc	- Buffer policy: FIFO - Buffer capacity: 1

COMPONENTE: interfaces	Características
Bus	- Concurrency: 1 - Arbitration Protocol: priority - Transfer Time: $\frac{USERDATASIZE}{\text{BusThroughput}}$
predictorControl	-

encoderControl	-

Tabla 15: CCSDS-123 CoFluent – Componentes *combinedSWHWPlatform*

PARÁMETRO	SIGNIFICADO	TIPO	VALORES	VALORES POR DEFECTO
NumCores	Número de núcleos para ejecución	Rango	[1,6]	1
coProcSpeedUp	Factor de aceleración	Rango	[1,6]	1

Tabla 16: CCSDS-123 CoFluent - Parámetros de diseño *combinedSWHWPlatform*

4.5. Mapeado del sistema

Una vez definidas las plataformas, el último paso en el modelado es el de asignación de funciones y variables a sus correspondientes componentes para la definición de la arquitectura. A continuación se explican los mapeados que se han llevado a cabo.

4.5.1. Mapeado de *simplePlatform*

En el primer modelo de arquitectura, se realiza una única asignación desde el modelo a la plataforma, al no haber más que un único componente presente.

En la Fig. 26 se observa la asignación de todas las funciones implicadas en el proceso de compresión al procesador *compressorProc*, de manera que estas funciones se ejecutarán de acuerdo a las características del planificador del procesador; al parametrizar el número de núcleos, se podrá paralelizar la ejecución en consecuencia, y observar cómo afecte este parámetro en posibles exploraciones.

En cuanto a las comunicaciones, al no haber distintos elementos desde y hacia los cuales comunicarse, se realizan de manera transparente e interna.

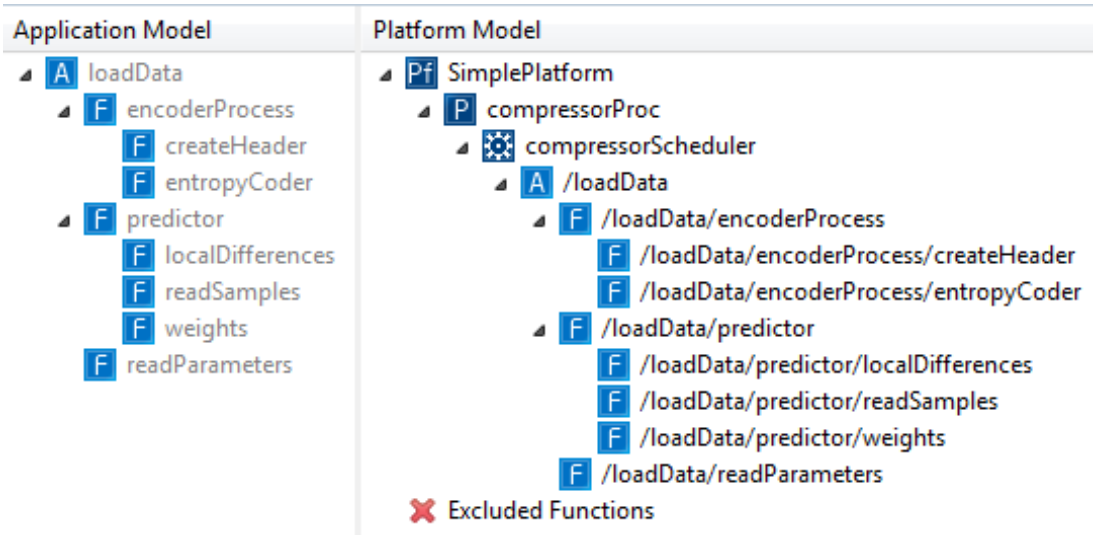

Fig. 26: CCSDS-123 CoFluent - *simplePlatformMapping*

4.5.2. Mapeado de combinedSWHWPlatform

En cuanto al segundo modelo de arquitectura, se cuenta con la segunda plataforma diseñada. El sistema en este caso presenta dos procesadores, cada uno de ellos agrupará distintos grupos de funciones.

En el primero de los procesadores, *predictorProc*, se asignan las funciones correspondientes a la predicción: *readSamples*, *localDifferences* y *weights*, sin que éste presente cambios respecto a la configuración de la plataforma *simplePlatform*. Sobre el segundo, *encoderProc*, se ejecutarán las funciones propias de la codificación, como si de un procesador específico hardware se tratase, al no contar con planificador. En Fig. 27 quedan reflejadas las asignaciones de funciones y rutas de datos.

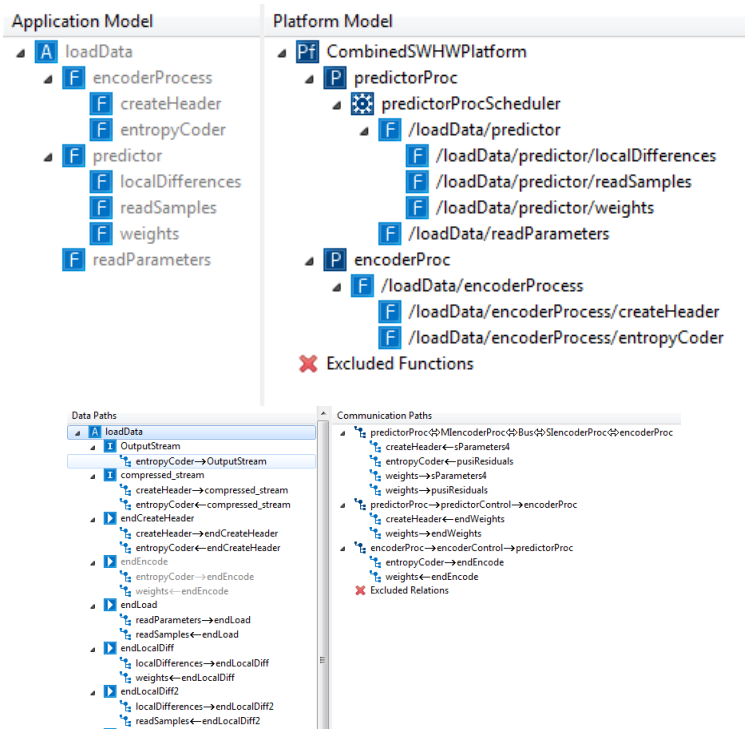

Fig. 27: CCSDS-123 CoFluent – *combinedPlatformMapping*

Se asignan las comunicaciones asociadas a las variables compartidas entre la función de predicción y la función de codificación, *sParameters4* y *pusiResiduals*, al bus de la plataforma. Los eventos de señalización *endWeights* y *endEncode*, se asignan a las conexiones simples *predictorControl* y *encoderControl*, respectivamente.

4.6. Resultados

A continuación se presentan los resultados que muestran el comportamiento de los modelos explicados anteriormente.

En primer lugar, se indican los resultados obtenidos para el modelo de aplicación, con la intención de mostrar el comportamiento obtenido de la aplicación modelada, y el flujo de trabajo asociado, para un caso ideal de ejecución paralela. En Fig. 28 se muestran los tiempos de ejecución de cada una de las funciones del proceso de compresión en porcentajes del tiempo total de ejecución. En la imagen se puede observar como la parte encargada de realizar las tareas de codificación es una de las que ocupa mayor tiempo del proceso, para un caso básico en el que se escogen tres bandas para la predicción, y una codificación *Sample Adaptive*. En concreto, la parte

correspondiente al cálculo de las predicciones y su mapeado, *weights*, también supone una parte del tiempo total considerable respecto al resto de tareas de predicción.

Function	RUNNING	PASSIVE	WAITING RES...
createHeader	0.9 %	0.0 %	97.8 %
encoderProcess	95.7 %	0.0 %	4.3 %
entropyCoder	94.9 %	0.0 %	5.1 %
localDifferences	34.5 %	0.0 %	61.0 %
predictor	57.0 %	0.0 %	41.7 %
readParameters	0.0 %	0.0 %	92.2 %
readSamples	39.0 %	0.0 %	53.2 %
weights	54.6 %	0.0 %	44.1 %

Fig. 28: CCSDS-123 CoFluent – Tiempos de ejecución de modelo de aplicación

La herramienta permite recoger de forma gráfica las ejecuciones a través de la representación de las señales y eventos a lo largo del tiempo, como la gráfica que se muestra en Fig. 29.

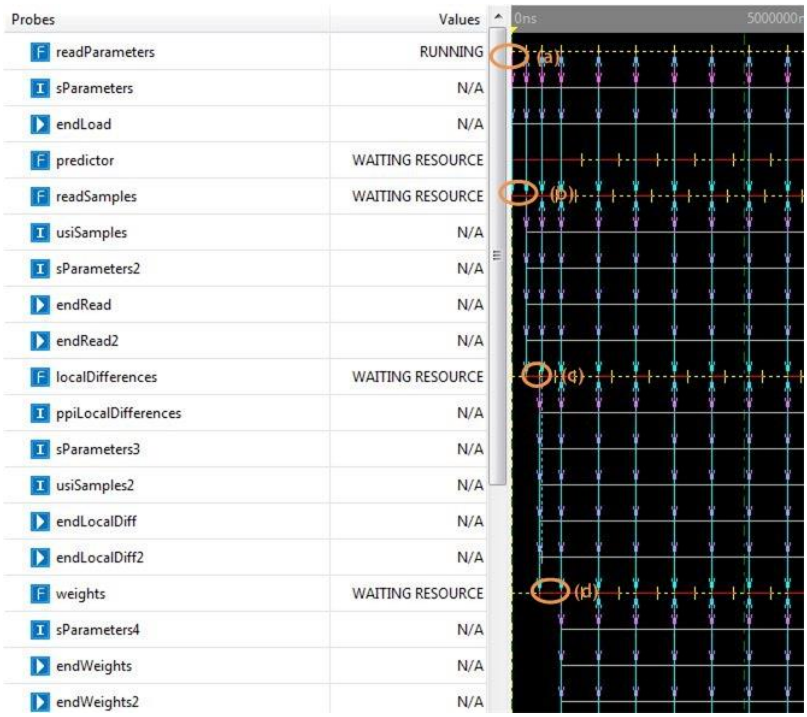

Fig. 29: CCSDS-123 CoFluent - *TimeLine* modelo de aplicación

En la figura se puede observar el correcto comportamiento, en cuanto a sincronización, que se ha pretendido proporcionar a través los eventos oportunos, y el flujo de ejecución, también se puede observar por la parte izquierda de la gráfica, cuándo se van activando por primera vez las distintas

funciones de las primeras etapas de compresión, indicado por la sucesión resaltada. Las líneas horizontales rojas se corresponden con periodos de actividad de cada función, mientras que las líneas discontinuas amarillas, se corresponden con periodos de inactividad, a la espera de algún determinado recurso.

Se desea aclarar que, las comprobaciones de eventos de etapas posteriores de compresiones anteriores, se realizan en el momento inmediatamente anterior a la propagación de las salidas de la compresión actual, de manera que estas salidas no se propagan hasta el comienzo del siguiente ciclo de ejecución; por lo que, aunque pueda parecer que no se producen resultados, éstos se generan y se propagan al comienzo del siguiente ciclo de ejecución.

En cuanto a tiempos, Fig. 30 muestra la latencia del modelo y Fig. 31 muestra el *throughput* estimado en función de los tiempos entre salidas y el número de muestras comprimidas, tal y como se definen en Ecuación 7 y en Ecuación 8; ambas figuras muestran los resultados para el caso de codificación *Sample Adaptive* y *Block Adaptive*. Las distintas gráficas que se muestran, representan los valores de latencia y *throughput* calculados en el eje y, durante las diferentes compresiones definidas por *Iterations* en el eje x. En este punto, es necesario explicar que la simulación del modelo de aplicación, paraleliza todas las funciones mientras sea posible. A partir de la premisa anterior, con el conocimiento del algoritmo, sus restricciones y la sincronización que se le ha dado al modelo, se entiende como la latencia aumenta inicialmente, al adelantar cálculos para posteriores compresiones, pero llega un momento en el que permanece constante, por la dependencia de datos entre sub-funciones, así como por la dependencia resultante de la propia sincronización.

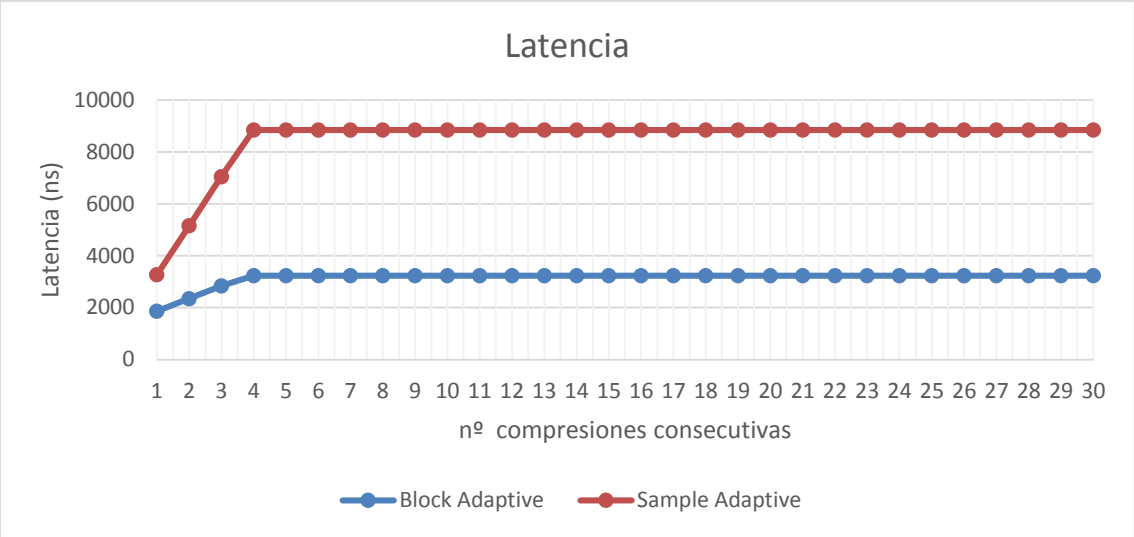

Fig. 30: CCSDS-123 CoFluent - Latencia de modelo de aplicación

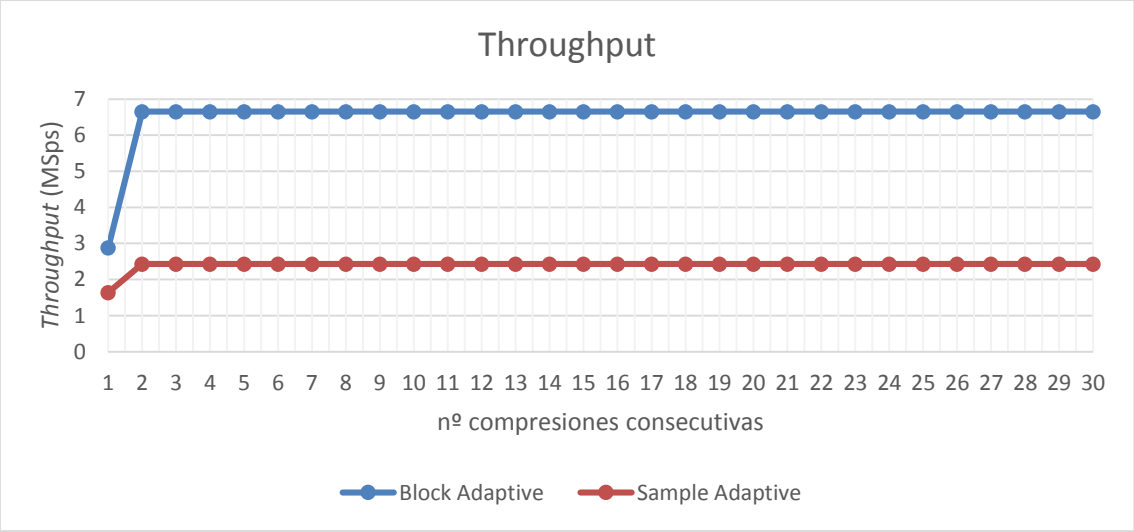

Fig. 31: CCSDS-123 CoFluent - *Throughput* de modelo de aplicación

Con la última gráfica se observa que las tasas de salida para distintas compresiones permanecen constantes por el mismo motivo explicado anteriormente.

El *throughput*, de acuerdo a los resultados obtenidos y representados, alcanza un valor de 6.65 MSps y 2.43 MSps para el caso de codificador entrópico *Block Adaptive* y codificador entrópico *Sample Adaptive*, respectivamente, lo cual viene justificado directamente por el tiempo asociado a cada proceso de codificación, de acuerdo a las estimaciones realizadas sobre el código de referencia, tal y como se explica en la sección

4.3.

Los resultados del modelo de aplicación sobre la primera plataforma se muestran a continuación.

Al contar con un único procesador, y un planificador, la ejecución paralela de funciones depende del número de núcleos asociados al planificador; por tanto es de esperar que los resultados de rendimiento estén directamente relacionados con este parámetro.

Fig. 32 muestra la gráfica temporal o *timeline* asociada a un planificador con un único núcleo, donde se observa cómo en la parte correspondiente a la predicción, las sub-funciones se van intercalando en ejecución en función de los eventos de señalización y la propia dependencia entre ellas, pero en ningún momento se ejecutan varias en paralelo. La parte resaltada de la imagen se corresponde con lo anteriormente descrito: para el proceso de predicción aparecen la etapa de lectura de muestras, la etapa del cálculo de diferencias locales y la etapa del cálculo de predicción y mapeado de residuos, sin solapamiento entre ellas.

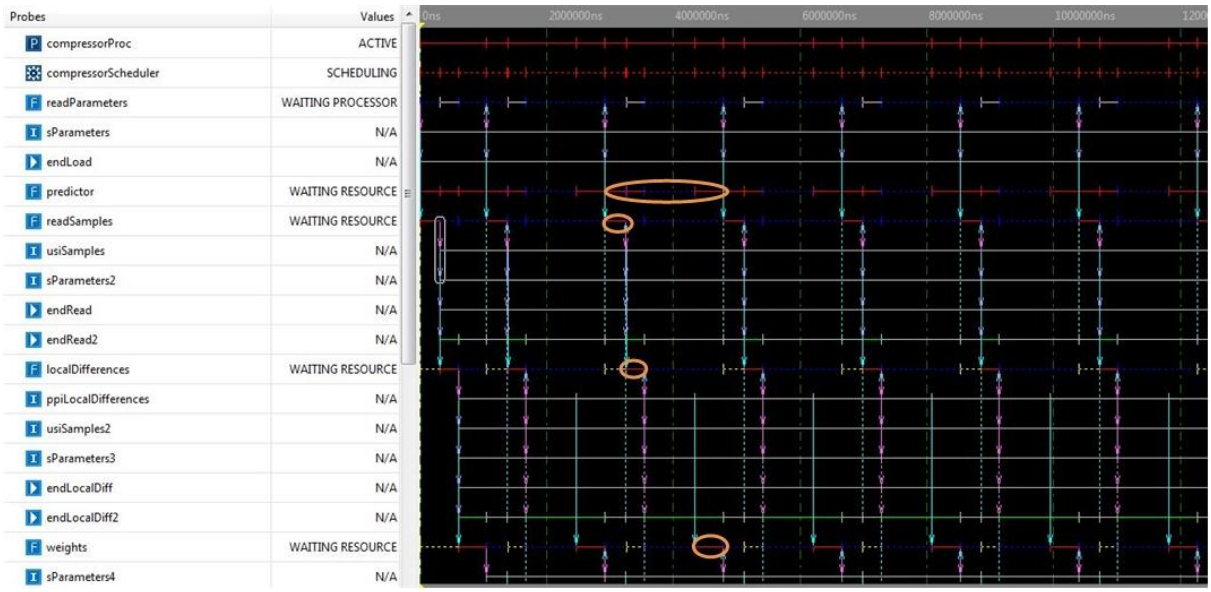

Fig. 32: CCSDS-123 CoFluent - *TimeLine simplePlatform* (1 core)

En Fig. 33, se aprecia mejor la dependencia de datos entre funciones, como la consecución marcada como *(a)*, *(b)* y *(c)*, ya que pese a contar ahora la simulación con tres núcleos, se observan momentos en los que el planificador no puede asignar tres funciones al mismo tiempo (lapso de

tiempo en el que sólo están ejecutándose *(d)* y *(e)*, ya que la primera de ellas es la que inicia (f) tras su finalización). Pese a que se ha intentado adelantar etapas previas en la medida de lo posible con el fin de mejorar los tiempos, ya que se recuerda que el propio código de referencia trabaja a nivel de imagen, la implementación no consigue librarse de este tipo de dependencias, y mejorar tiempos a fuerza de aumentar los núcleos del procesador.

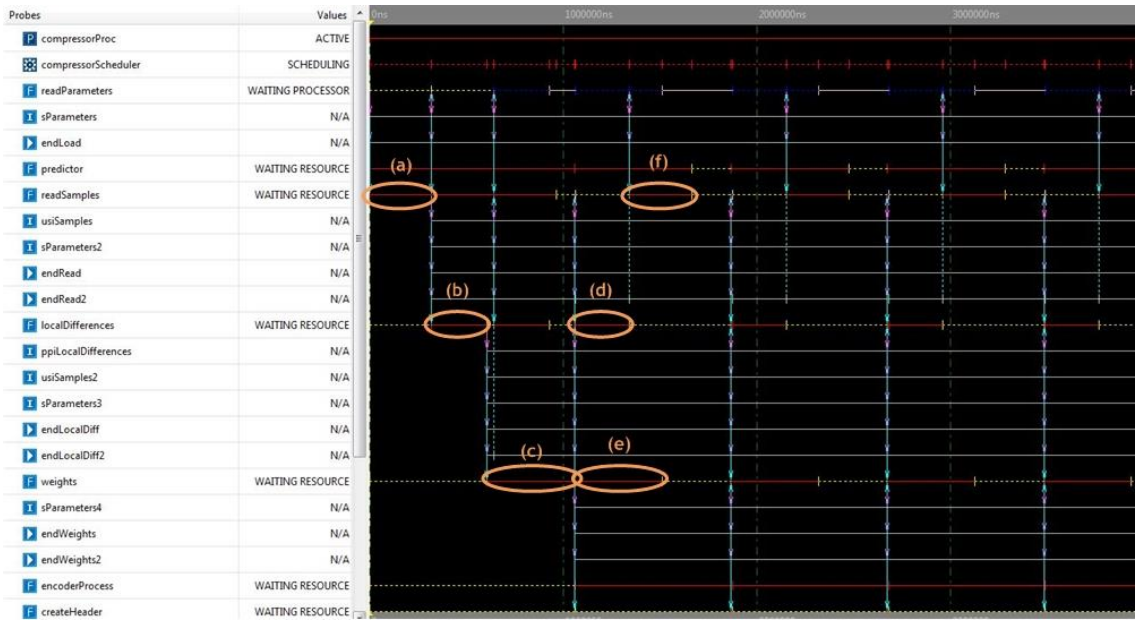

Fig. 33: CCSDS-123 CoFluent - *TimeLine simplePlatform* (3 cores)

Las figuras que se muestran a continuación respaldan las dependencias mencionadas. Fig. 34 y Fig. 36 muestran las latencias para cada tipo de compresión y en función del número de núcleos de procesamiento. En las gráficas se observa un aumento inicial que finalmente se estabiliza para cada número de núcleos asociados, y también se observa cómo a medida que aumentan los núcleos, con lo que se pueden adelantar sub-funciones, se produce un aumento de latencia. Fig. 35 y Fig. 37 muestran el *throughput* también para cada caso de codificación, y los valores correspondientes al número de núcleos asociados. En este caso hay una mayor diferenciación a parte de los propios valores obtenidos. Al contar con el codificador *Sample Adaptive*, el tiempo de codificación asociado impone una restricción mayor que en el caso del codificador *Block Adaptive*, siendo tal la restricción que en el caso del primer codificador, no supone mejora en el *throughput* a partir de

dos núcleos, mientras que en el segundo hay diferencia entre dos y tres, dejando de suponer mejora a partir del tercer núcleo. En ningún caso se puede llegar a superar las tasas del modelo de aplicación, el cual se recuerda que considera paralelización máxima considerando como único límite las dependencias.

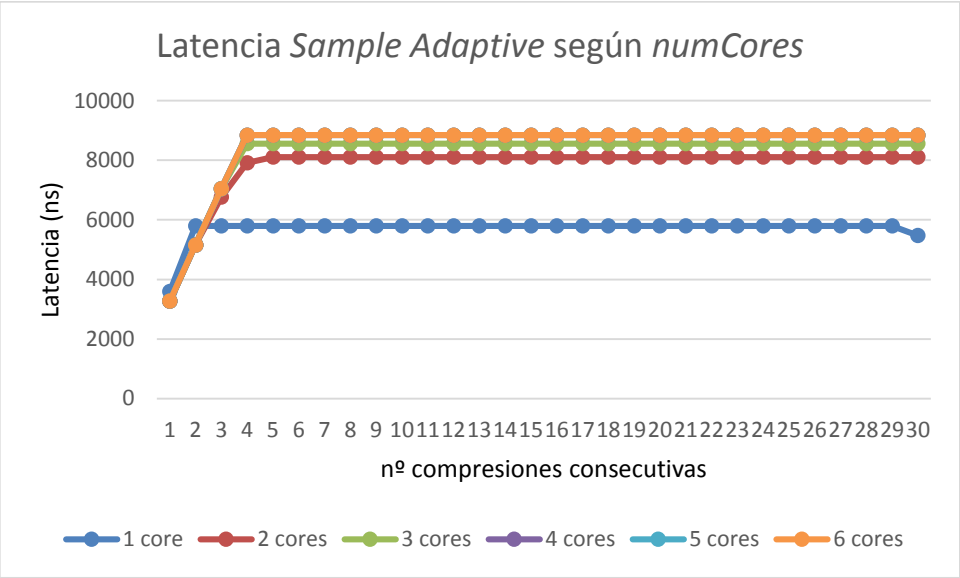

Fig. 34: CCSDS-123 CoFluent - Latencia de *simplePlatform* según *numCores* (*Sample Adaptive*)

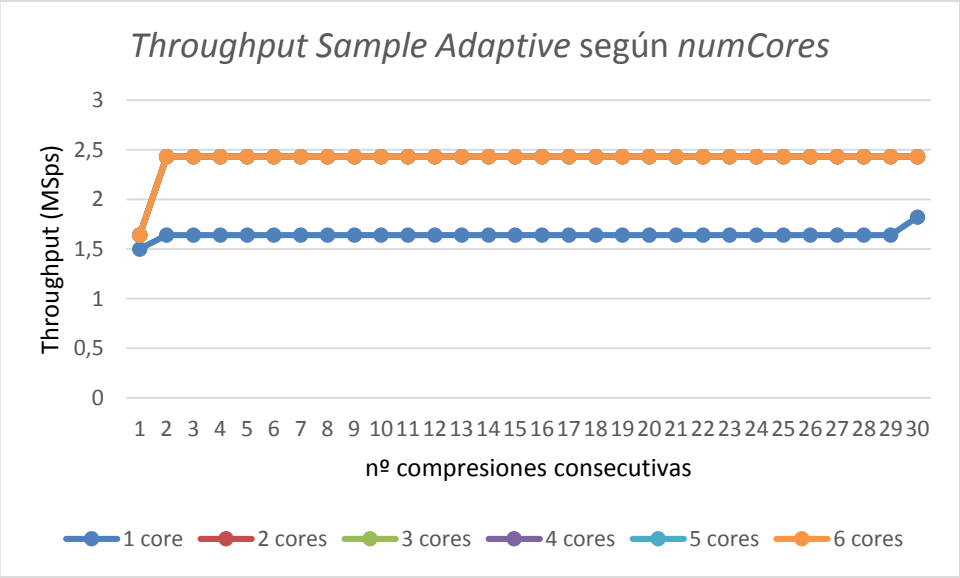

Fig. 35: CCSDS-123 CoFluent - *Throughput* de *simplePlatform* según *NumCores* (*Sample Adaptive*)

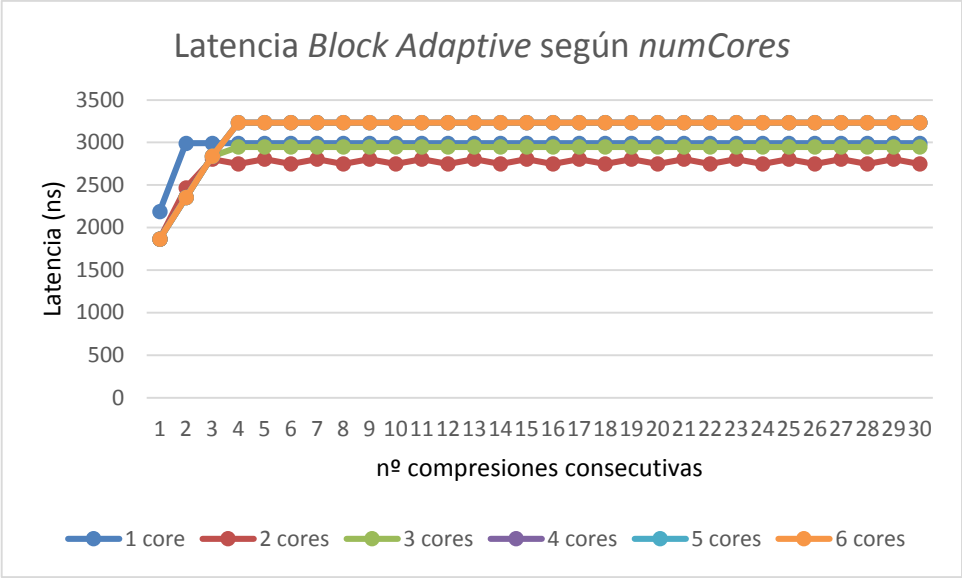

Fig. 36: CCSDS-123 CoFluent - Latencia de *simplePlatform* según *numCores* (*Block Adaptive*)

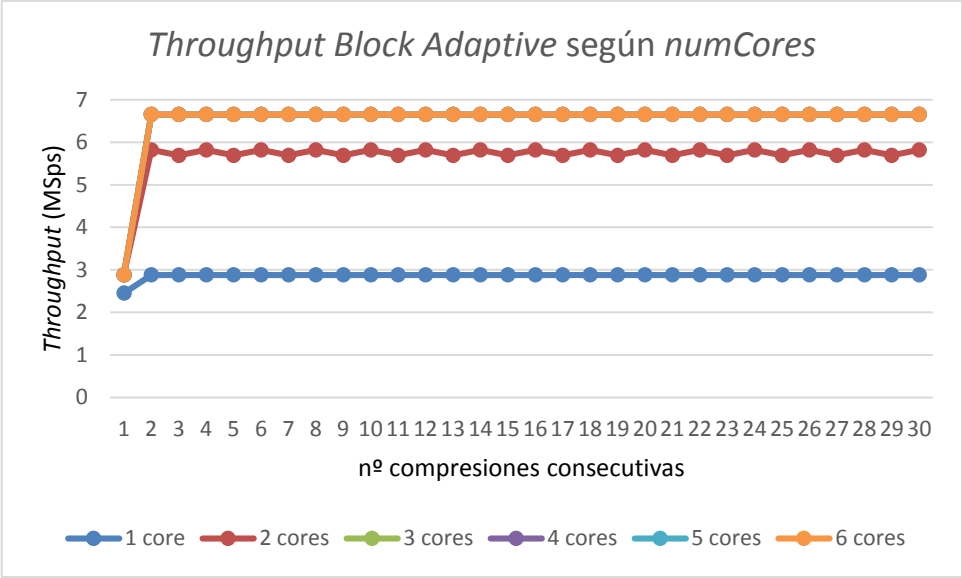

Fig. 37: CCSDS-123 CoFluent - *Throughput* de simplePlatform según *numCores* (*Block Adaptive*)

Por último se analizan los resultados de *combinedSWHWPlatform*, donde la idea, es mejorar los tiempos mediante la aceleración de la parte correspondiente a la codificación. En

Fig. 38 se aprecian dos claros efectos de esta plataforma. Por un lado, se introduce un nuevo retardo en el sistema al considerar en este caso la comunicación de la imagen por el bus, elemento que puede y seguramente estará en cualquier sistema de estas características, resaltado como *(a)* en la

imagen, que obviamente es un retardo que depende del tamaño de la imagen y de la capacidad de transmisión del bus; por otro lado se influye en el tiempo de ejecución asociado a la codificación entrópica, resaltado en la imagen como *(b)*, mediante el parámetro *encoderProcSpeedUp*, que se recuerda es parámetro de la plataforma.

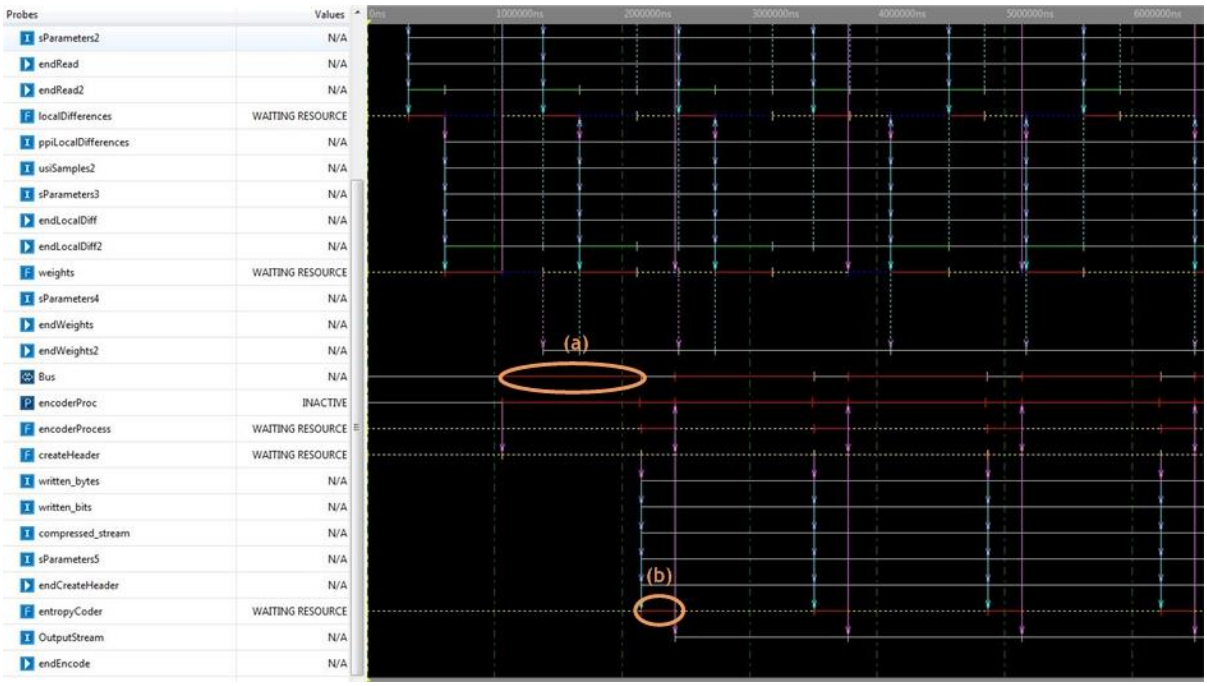

Fig. 38: CCSDS-123 CoFluent - TimeLine *combinedSWHWPlatform*

Por tanto, teniendo en cuenta la condición de transmisión por el bus, el sentido real de esta arquitectura no es la de comparar tiempos con la arquitectura anterior, ya que como se ha especificado por la incorporación del bus no son comparables, sino establecer medidas de la aceleración de codificación en función del factor de aceleración del procesador correspondiente.

Fig. 39, Fig. 41, Fig. 40 y Fig. 42 muestran la latencia y el *throughput* de la aplicación, respectivamente, para distintos valores del factor de aceleración; en estos casos, la disminución de latencia y el aumento de la tasa de salida, llegan a unos valores determinados, a partir de los cuales las variaciones son cada vez menos significativas pese a que aumente el factor de aceleración asociado a la parte de codificación.

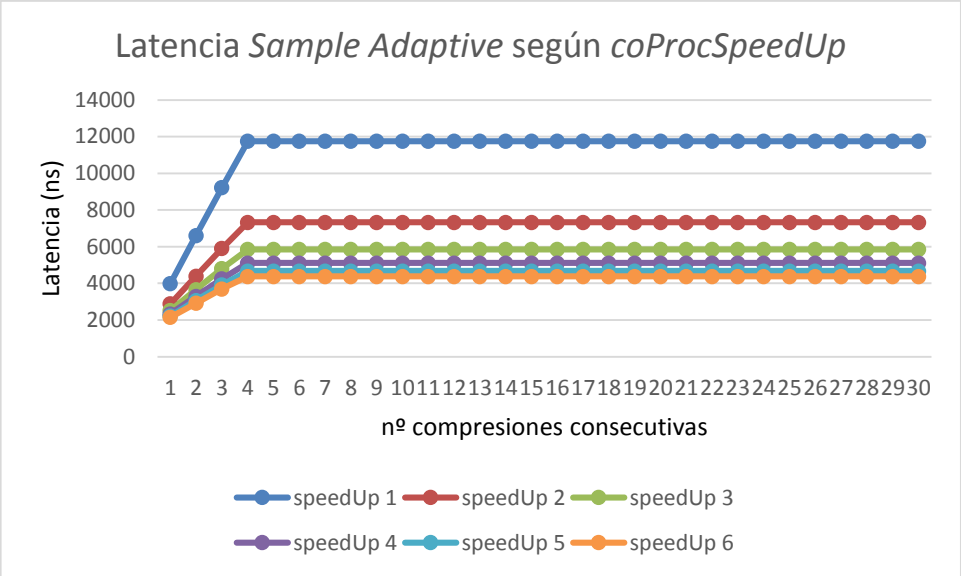

Fig. 39: CCSDS-123 CoFluent - Latencia de *combinedSWHWPlatform* según *encoderProcSpeedUp* (*Sample Adaptive*)

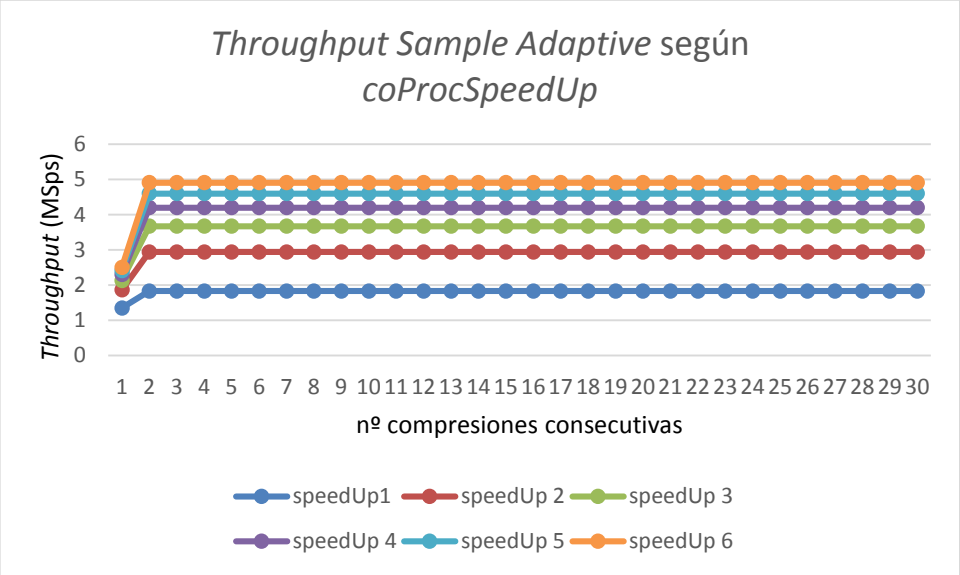

Fig. 40: CCSDS-123 CoFluent - *Throughput* de *combinedSWHWPlatform* según *encoderProcSpeedUp* (*Sample Adaptive*)

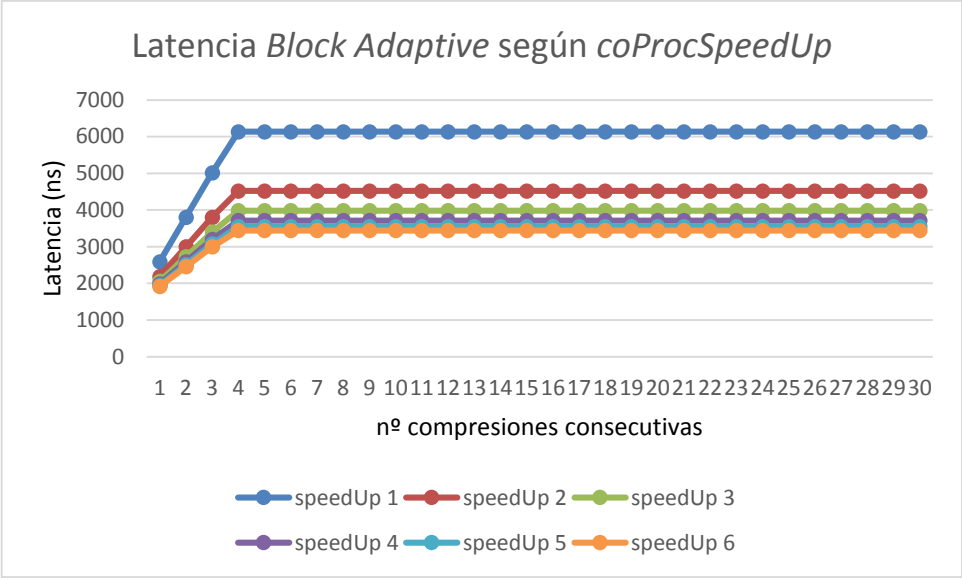

Fig. 41: CCSDS-123 CoFluent - Latencia de *combinedSWHWPlatform* según *encoderProcSpeedUp* (*Block Adaptive*)

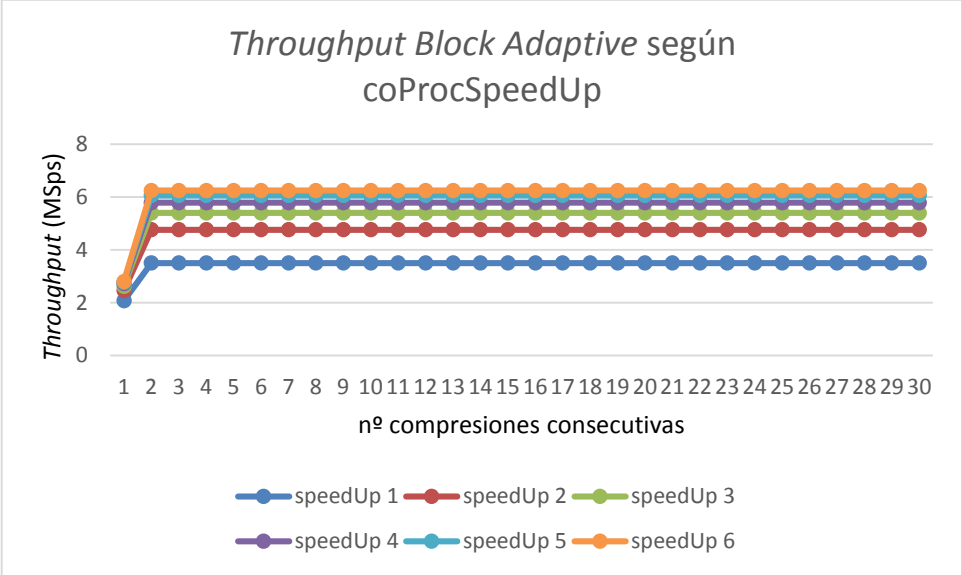

Fig. 42: CCSDS-123 CoFluent - *Throughput* de *combinedSWHWPlatform* según *encoderProcSpeedUp* (*Block Adaptive*)

Por tanto, una vez establecida la existencia de un límite en cuanto a los tiempos obtenidos de relativos a la parte del proceso de codificación, se vuelve a realizar una exploración general, esta vez en función del número de bandas utilizadas en la predicción.

Partiendo del modelo de aplicación inicial y sin tener en cuenta bus ni aceleración de ningún tipo, se ha estudiado la influencia del parámetro

predBands. Fig. 43 y Fig. 44 muestran los resultados obtenidos en la exploración del modelo de aplicación, en los que se puede apreciar el impacto de este parámetro de bandas previas a considerar en la predicción. Puede llegar a afectar hasta tal punto que el *throughput* llega a decrementarse en un factor superior a 2.5 a medida que se incluyen bandas a considerar para la predicción, hasta llegar a las 15 permitidas por el estándar. Se considera necesario también mejorar los tiempos en general, así como los tiempos de predicción, para los casos particulares de número elevado de bandas. Esta restricción del número de bandas afectará en mayor o menor medida en función una serie de factores, como por ejemplo el tamaño de la imagen o la codificación elegida, ya que este tiempo se considera únicamente en uno de los procesos de la predicción. Para el caso concreto de las imágenes, se ha realizado con simulaciones sobre una imagen de tamaño $16 \times 16 \times 21$ muestras y para una codificación *Block Adaptive*.

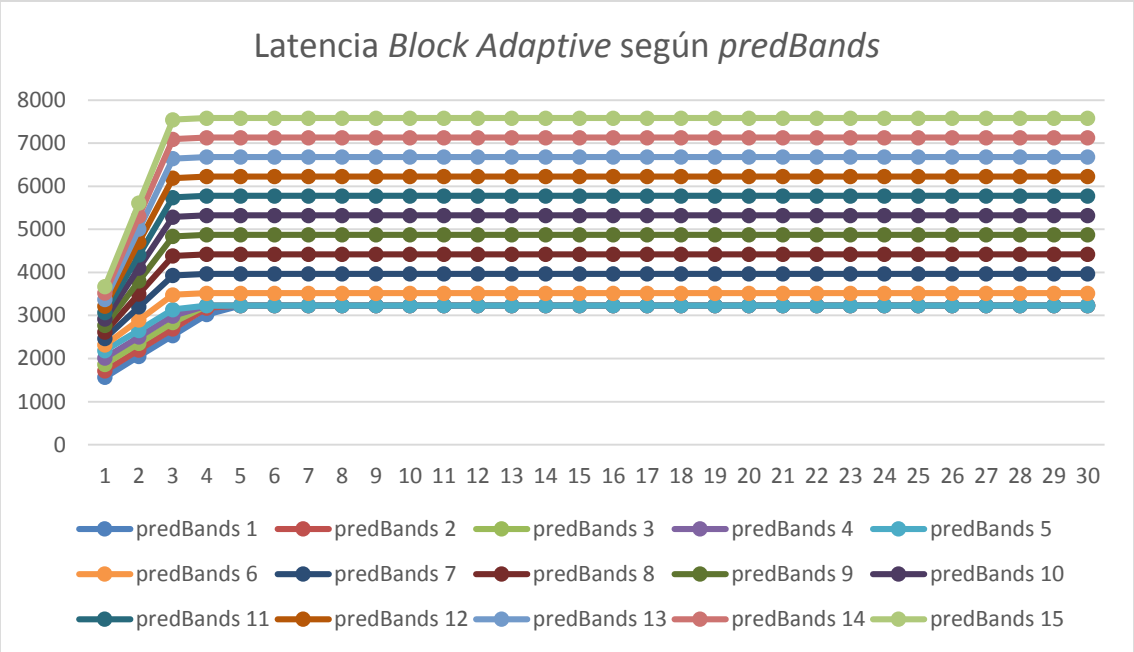

Fig. 43: CCSDS-123 CCSDS-123 CoFluent - Latencia del modelo de aplicación según *predBands* (*Block Adaptive*)

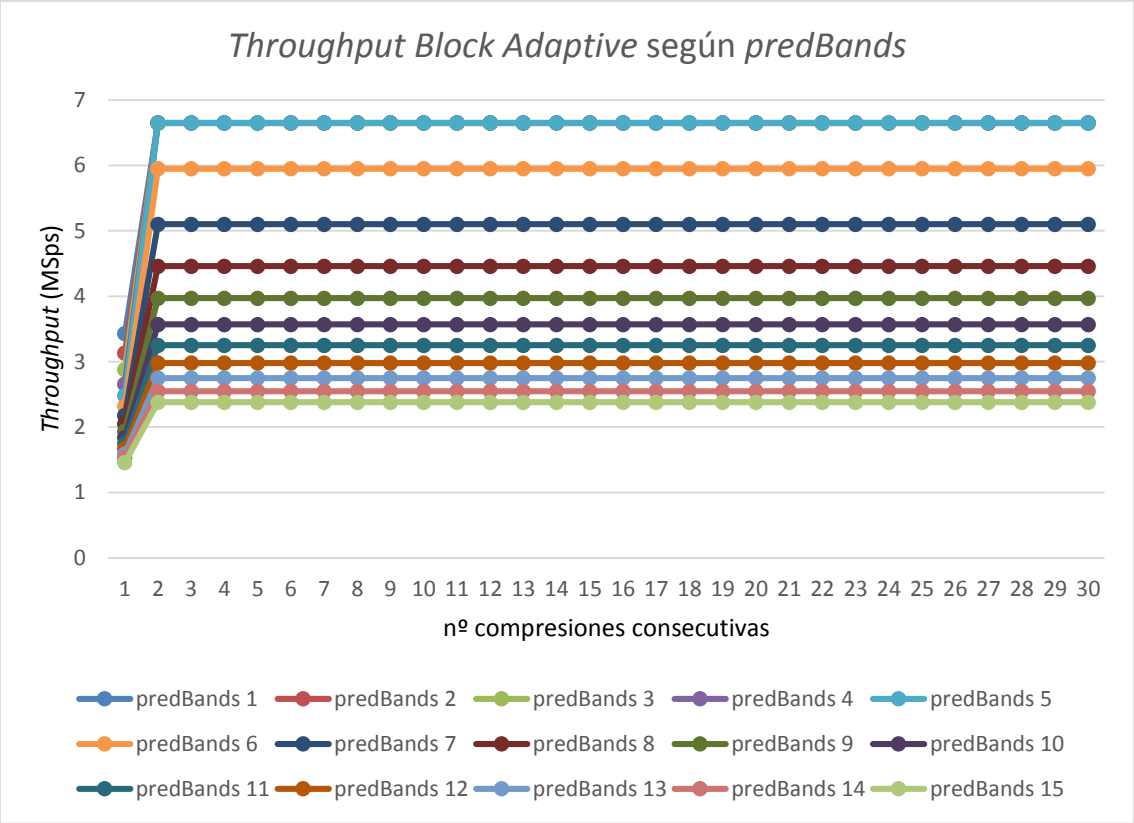

Fig. 44: CCSDS-123 CCSDS-123 CoFluent - *Throughput* del modelo de aplicación según *predBands* (*Block Adaptive*)

5. Optimización del modelo de referencia

5.1. Modelo optimizado

A partir de los resultados obtenidos en las simulaciones del modelo anterior, se observa un bajo nivel de paralelismo del algoritmo, inherente a éste y no al propio estándar, que puede llegar a provocar retrasos considerables cuando las imágenes de entrada son de tamaño considerable, e incluso en función de determinados parámetros, como puede ser el tamaño de bloque para codificación con *Block Adaptive*, o según el número de bandas usadas para la predicción. Con la intención de mejorar el tiempo de ejecución del algoritmo, se ha realizado un nuevo modelo basado en el primero, que permite trabajar con los residuos mapeados tan pronto hayan sido calculados en la función de predicción (el modelo original espera a terminar de computar los todos los residuos para realizar la codificación) consiguiendo con ello mejorar el tiempo final de ejecución, adelantando en la medida de lo posible las tareas de codificación.

Para poder realizar las modificaciones necesarias, se ha tenido que llevar algunas partes algorítmicas que inicialmente se incluyeron como fuente externa, al editor gráfico de la herramienta, y modificar partes de los propios algoritmos para la nueva descripción, al mismo tiempo que se ha añadido código de control al modelo mediante la incorporación de distintos elementos al diseño.

La aceleración a priori no es máxima, por el propio modo de trabajo con el cubo de datos, como se detalla a continuación. Pese a que el estándar permite como imágenes de entrada imágenes con formatos de entrada BIP y BIL, la lectura de muestras reorganiza la matriz en el propio sistema, y siempre se realiza la parte de predicción en orden de acuerdo al formato BSQ, por lo que para el caso BIL o BIP, la salida de residuos individuales no proporciona una codificación inmediata en la siguiente etapa de procesamiento.

Al paralelizar los procesos, y para evitar problemas de sincronización de cada función, se ha implementado una cola de mensajes entre el predictor y

el codificador, para que los residuos se almacenen en caso de que no poder ser debidamente recibidos por la etapa de codificación inmediatamente después de haber sido generados en la etapa de predicción.

A continuación, se muestra en Fig. 45 se muestra una representación del nuevo modelo. Las distintas sub-funciones aparecen encapsuladas, por simplicidad a la hora de trabajar, así como se ha reestructurado su descripción gráfica, pero el comportamiento de aquellas funciones no implicadas en la transferencia y tratamiento directo de residuos, es el mismo que en el modelo de aplicación inicial. La cola de residuos, que en la figura aparece resaltada, es la nueva variable compartida en la que la sub-función *weights* deposita los residuos individuales y mapeados de cada muestra, para su posterior procesamiento en la sub-función de codificación.

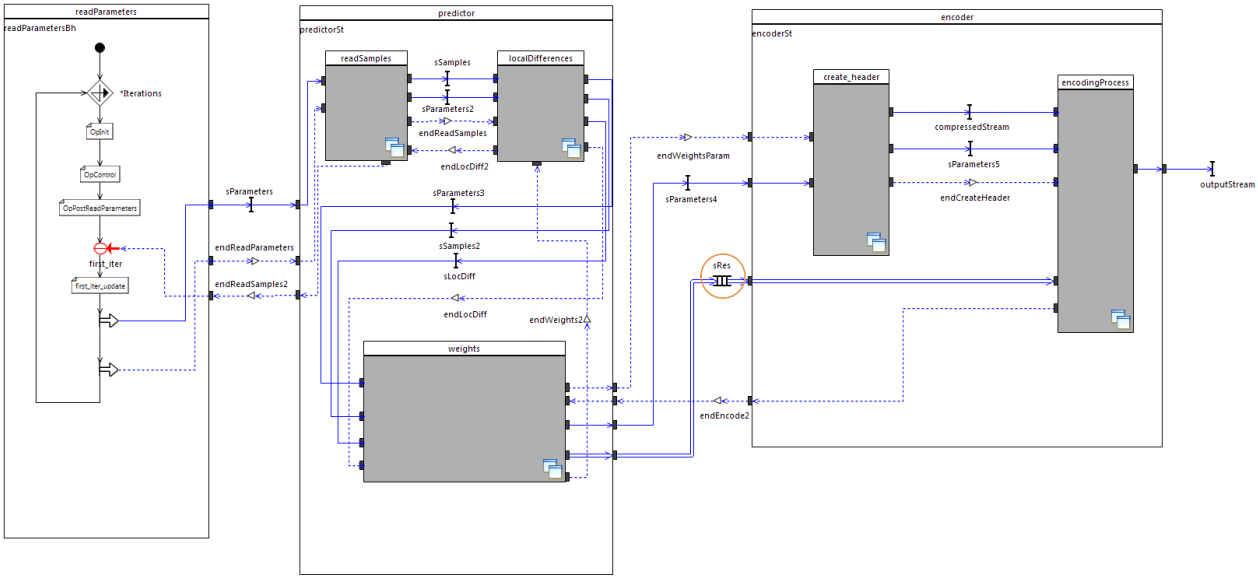

Fig. 45: CCSDS-123 CoFluent - Mejora de modelo de aplicación

Las funciones de cálculo de residuos y codificación, se han modificado para permitir tratar cada uno de los residuos tan pronto como estén calculados; la sub-función *weights*, encargada del cálculo y mapeado de los residuos, queda como se muestra en Fig. 46. El evento que activa la parte de cabecera de la siguiente etapa, *endWeightsParam*, se activa con anterioridad al cálculo de cualquier residuo, para adelantar también la ejecución de la formación de cabecera siempre que sea posible. A continuación y a través de tres bucles que controlan la posición de la muestra a calcular en la matriz tridimensional

de datos, se calculan y se envían uno a uno los residuos a la parte correspondiente de codificación de la siguiente etapa. Los tiempos se han recalculado a nivel de residuos y se muestran en Tabla 17 con los detalles de los componentes.

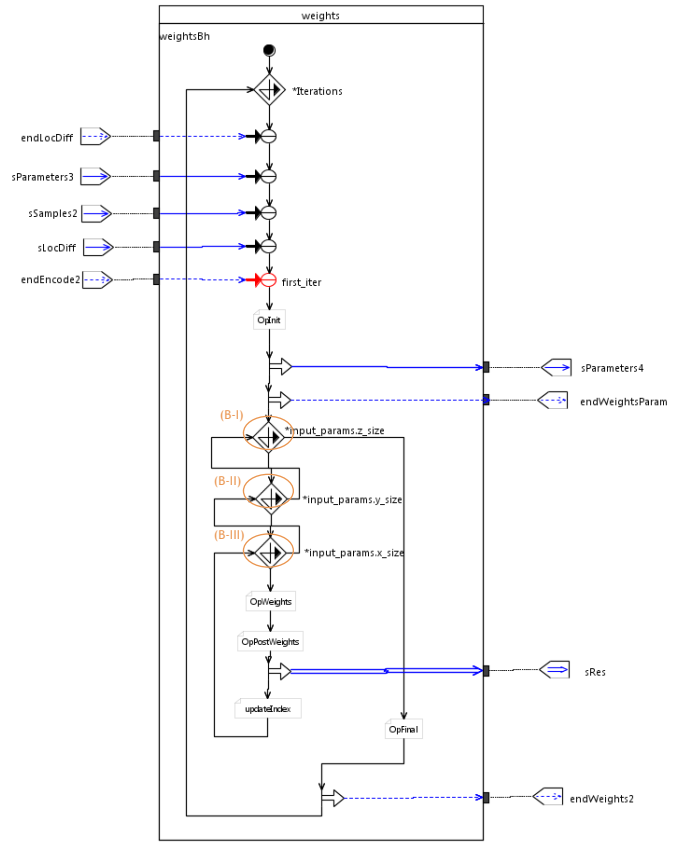

Fig. 46: CCSDS-123 CoFluent – función *weights* mejorada

COMPONENTE: Bucle	TIPO
Bucle B-I	(*for*) {*input_params.z_size*}
Bucle B-II	(*for*) {*input_params.y_size*}
Bucle B-III	(*for*) {*input_params.x_size*}
COMPONENTE: Operaciones	**TIEMPO DE EJECUCIÓN (ns)**
OpInit	10
OpWeights	*fWeightsTime*
OpPostWeights	*1*
updateIndex	1
OpFinal	1

Tabla 17: CCSDS-123 CoFluent - Componentes de función *weights* mejorada

$$fWeightsTime = 27 \times predictor\_params.pred\_bands$$

La siguiente sub-función modificada de este modelo, queda reflejada en Fig. 47. Se trata de una primera parte correspondiente al control de la posición inmediata de residuo; si bien no es estrictamente necesaria para la recepción de los residuos, pues a priori en este punto del algoritmo se saben cuántos residuos deben llegar, se implementa porque permite establecer unos valores de control para la segunda etapa de esta misma función, y discernir cuándo se debe codificar por características y cuándo no, en función de estos mismos valores y de los parámetros de codificación. A continuación hay un almacenamiento del residuo previo a su tratamiento; este residuo irá bien a una estructura interna almacenado en su posición correcta, porque no se va a usar inmediatamente, o bien a una variable de tipo adecuado, ya que se va a tratar inmediatamente a continuación.

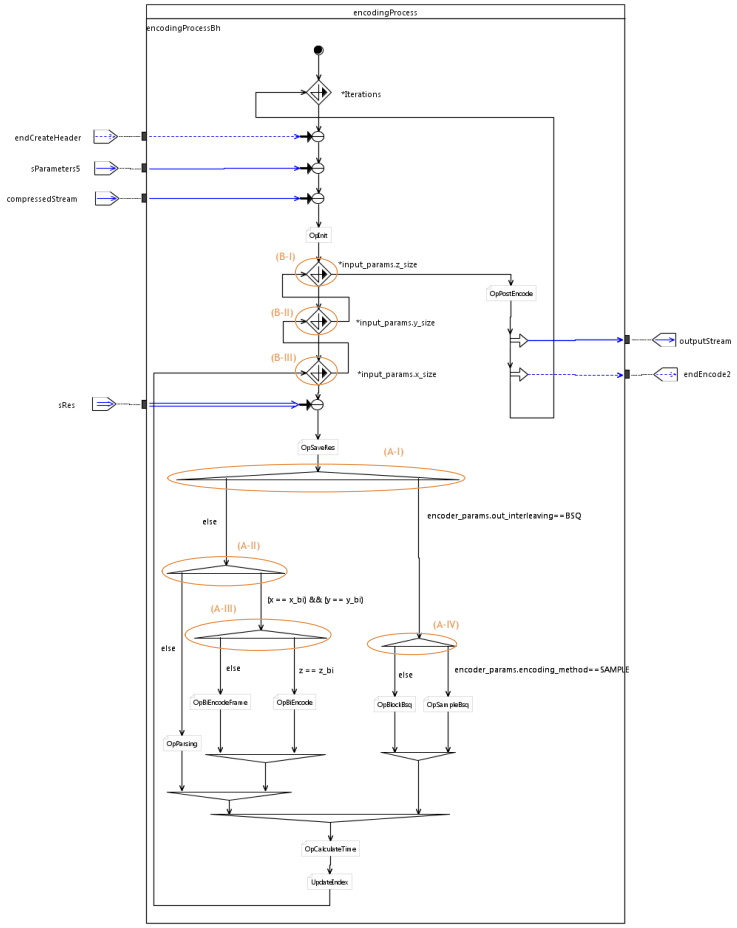

Fig. 47: CCSDS-123 CoFluent – función *encodingProcess* mejorada

Las características de los distintos componentes se muestran a continuación en Tabla 18. Se considera necesario aclarar en este punto los tiempos de las operaciones de esta sub-función. Si bien es posible establecer por la posición de llegada de cada residuo si se va a codificar inmediatamente o no, no es posible determinar cuántos residuos se van a codificar en cada instante de los que ya se hayan guardado de recepciones anteriores; por este motivo, se ha descrito una operación posterior al paso de codificación, OpCalculateTime, que calcula la cantidad de residuos codificados y establece un tiempo de codificación para ese paso, que finalmente se asocia a UpdateIndex. El tiempo final de codificación es análogo al modelo inicial, aunque se haya introducido dichos arreglos.

COMPONENTE: Bucle	**TIPO**
Bucle B-I	(*for*) {*input_params.z_size*}
Bucle B-II	(*for*) {*input_params.y_size*}
Bucle B-III	(*for*) {*input_params.x_size*}
COMPONENTE: Alternativa	**CONDICIÓN**
Alternativa A-I	Disposición de muestras: BSQ|BI $\{encoder\_params.out\_interleaving == BSQ\}$
Alternativa A-II	Muestra a considerar en formato BI $\{x == x_{bi} \,\&\&\, y == y_{bi}\}$
Alternativa A-III	Muestra a tratar inmediatamente o tras tratamiento de muestras almacenadas $\{z == z_{bi}\}$
Alternativa A-IV	Método de codificación: Sample|Block $\{encoder\_params.encoding\_method == SAMPLE\}$

COMPONENTE: Operaciones	TIEMPO DE EJECUCIÓN (ns)
OpInit	10
OpSaveRes	1
OpParsing	*1*
OpBiEncodeFrame	*1*
OpBiEncode	1
OpBlockBsq	1
OpSampleBSQ	1
OpCalculateTime	1
UpdateIndex	*fEncodeTime*
OpPostEncode	*1*

Tabla 18: CCSDS-123 CoFluent - Componentes de función *encodingProcess* mejorada

$fEncodeTime$ se calcula para cada llegada de residuo, en función del tratamiento que dicho residuo reciba, y del número de residuos guardados que se puedan tratar previamente.

5.2. Resultados

A continuación se muestran los resultados del nuevo modelo de aplicación. La primera imagen, Fig. 48, muestra el *TimeLine* del nuevo modelo, donde se puede observar las distintas transacciones de residuos para generar finalmente el *bitstream* de salida en cada ejecución del algoritmo (partes resaltadas en azul y rojo, respectivamente). El comportamiento es el esperado para las distintas posibilidades de predicción y codificación, y a continuación se muestra un ejemplo.

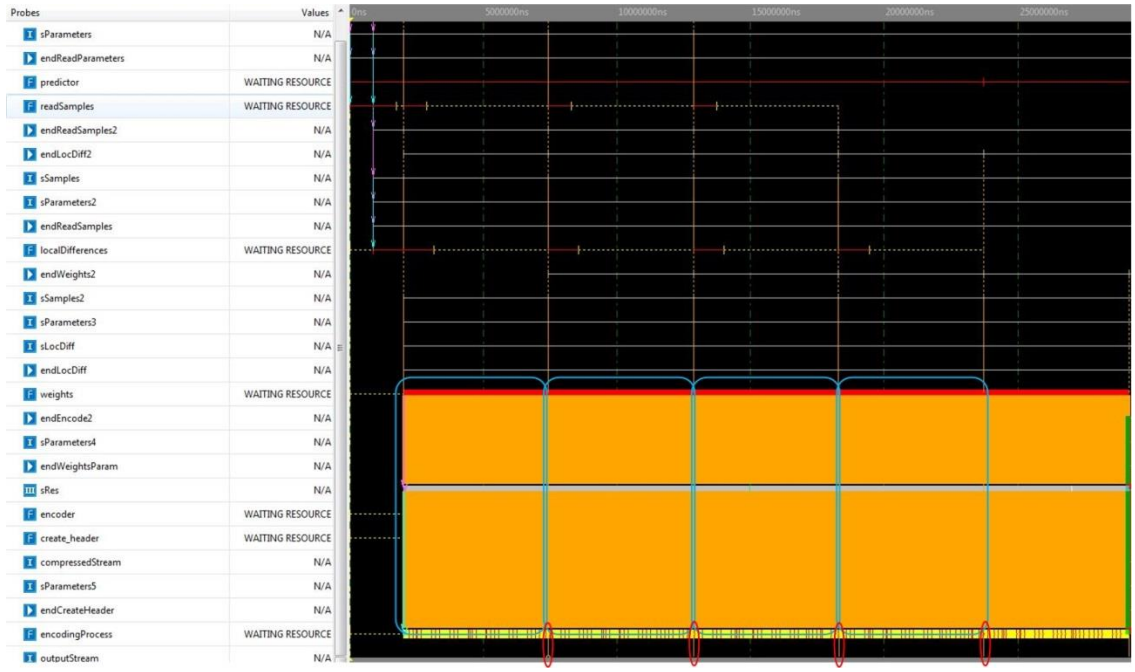

Fig. 48: CCSDS-123 CoFluent - *TimeLine* de modelo de aplicación mejorada

Fig. 49 muestra una parte de un ejemplo de codificación de imagen en formato BSQ con codificador Block de tamaño de bloque 16. Los primeros residuos se van trasfiriendo a la cola de residuos, *(a)*, pese a que la etapa de codificación no puede ir recogiéndolos al estar generando la cabecera, *(b)*, del bitstream de salida. Una vez finalizada dicha tarea, al codificador le llega el evento de finalización de cabecera y comienza a leer los residuos almacenados en la cola, *(c)*, hasta que no quedan residuos ya que todavía no se han generado nuevos. A partir de ese momento, los residuos se leen de la cola a medida que se van depositando en ella, *(d)*. También se resalta el momento concreto en el que se codifica un bloque de muestras, que coincide con el número de muestra 16 en este caso de almacenamiento *BSQ*, al haber alcanzado el tamaño indicado por el parámetro correspondiente, *(e)*.

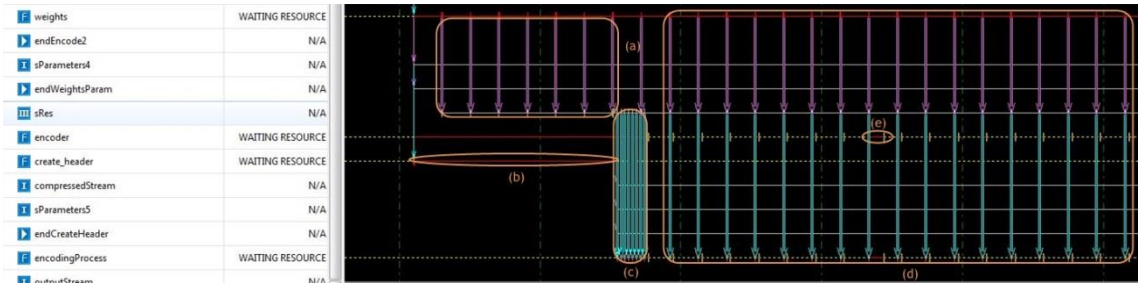

Fig. 49: CCSDS-123 CoFluent - *TimeLine* de codificación *Block-BSQ*

La verificación de la mejora del modelo inicial, pasa por realizar una exploración de los tiempos obtenidos en función del número de bandas usadas en predicción.

Fig. 50 permite comparar el *throughput* entre los modelos desarrollados, en los cuales se aprecia una ligera diferencia en la evolución de la tasa de salida. Si bien se esperaba una mejora más notable en el nuevo modelo, hay que apreciar una diferencia para el menor y mayor número de bandas previas en consideración para la predicción, que supone un procesamiento de 150000 muestras y 270000 muestras, respectivamente, por segundo, para el ejemplo ilustrado en concreto.

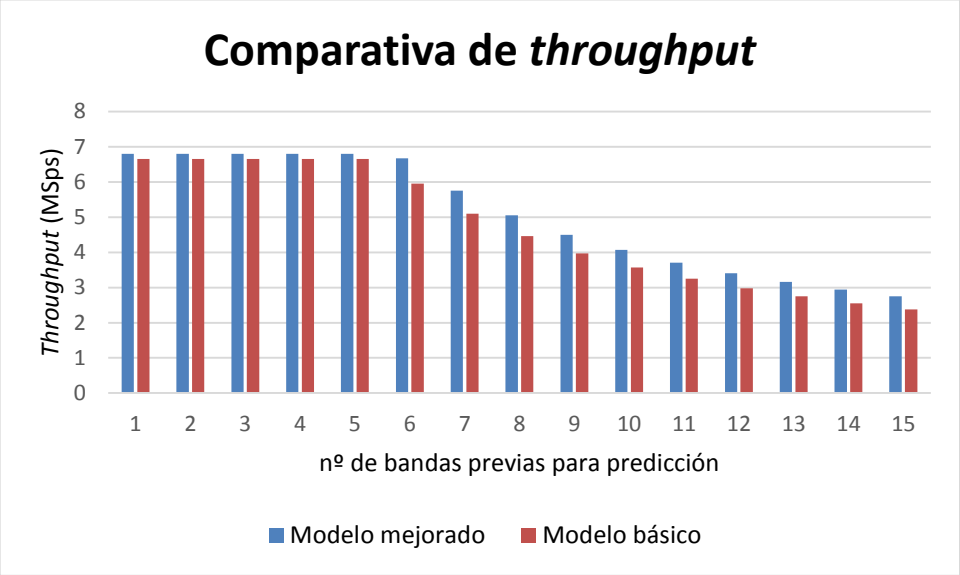

Fig. 50: CCSDS-123 CoFluent - *Throughput* Modelo mejorado / Modelo inicial

6. Conclusiones y trabajo futuro

Con este Trabajo Fin de Máster se ha modelado el algoritmo de referencia del estándar CCSDS-123, proporcionado por la ESA en la herramienta Intel CoFluent Studio. Este modelo, el cual se ha verificado directamente con el propio software de compresión, permite realizar diversas simulaciones para observar el impacto de determinados parámetros sobre los resultados en cuanto a temporalidad del propio algoritmo.

Para un caso general de pocas bandas de predicción (en torno al valor típico, $P = 3$), se ha observado que los resultados en tiempo se encuentran limitados por la parte del proceso correspondiente a la codificación, en especial para el caso de codificación *Sample Adaptive*, lo que se puede mejorar con una consideración de aceleración hardware. Esta mejora se ha demostrado con la segunda plataforma diseñada. Igualmente, se podrían volcar todas las funciones a un procesador con aceleración y obtener mejores resultados para el proceso en general.

Para mejorar tiempos tanto para el tratamiento de imágenes grandes como para reducir el efecto de dependencia en la predicción, y más cuando se cuenta con un número elevado de bandas previas para la etapa de predicción, se ha realizado un segundo modelo de aplicación que permite realizar los pasos de predicción y codificación a nivel de muestras. Con las modificaciones realizadas, se observa una mejora de tiempos de ejecución, aprovechando la disponibilidad de recursos y la ejecución paralela de distintas tareas, llegando a obtener una tasa de hasta 270000 muestras más por segundo que la resultante en la versión original, para el caso de mayor número de bandas usadas para la predicción.

A su vez este trabajo, que supone una primera aproximación al modelado del algoritmo de referencia del estándar, y se enmarca en el proyecto ya nombrado anteriormente, ITT No. AO/1-8032/14/NL/AK – CCSDS Lossless Compression IP-Core Space Applications, plantean las líneas futuras que se explican a continuación. Desarrollo de plataformas concretas y de mayor complejidad, donde poder mapear el modelo del compresor; refinamiento de

la caracterización considerando más variables del sistema, y conseguir una exploración del espacio de diseño que sea capaz de obtener datos no sólo en cuanto a tiempos, si no también caracterizar y cuantificar otros elementos como memoria, como paso previo a una posible implementación hardware del compresor, o bien obtener un modelo en *SystemC* equiparable a una implementación final hardware.

7. Listado de acrónimos

ACRÓNIMOS	
2D/3D-CALIC	2D/3D-Context-based, Adaptative, Lossless Image Codec
BI	Band-Interleaved
BIL	Band-Interleaved by Line
BIP	Band-Interleaved by Pixel
bpppb	bits per pixel per band
BSQ	Band-SeQuential
CCSDS	Consultative Committee for Space Data Systems
CDT	Transformada Discreta del Coseno
CR	Compression Ratio
ESA	European Space Agency
GOES	Geostationary Operational Environmental Satellite
HyspIRI	Hyperspectral Infrared Imager
IRS	Indian Remote Sensing
JPEG-LS	Lossless Joint Photographic Experts Group
LCL-3D	LCL-3D
LOCO-I	LOw COmplexity LOssless COmpression for Images
MSps	Mega-muestras por segundo
NASA	NASA
NOOA	National Oceanic and Atmospheric Administration
TBM	Timed-Behavioral Modeling
WDT	Transformada Discreta del Wavelet

8. Referencias

[1] *Lossless Multispectral & Hyperspectral Image Compression*. Recommendation for Space Data System Standards, CCSDS 123.0-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, May 2012.

[2] "CoFluent_guess or Model" [online:] http://www.cofluentdesign.com [Última consulta: Abril de 2015].

[3] IntelCoFluentStudio_v5.2_Reference_Guide, Intel Corporation.

[4] IntelCoFluentStudio_v5.2_Users_Guide, Intel Corporation.

[5] IntelCoFluentStudio_v5.2_Methodology_Guide, Intel Corporation.

[6] Appendix 1 to ESA ITT AO/1-8032/14/NL/AK, Statement of Work, CCSDS Lossless Compression IP-Cores.

[7] *Lossless Data Compression*. Recommendation for Space Data System Standards, CCSDS 121.0-B-2. Blue Book. Issue 2. Washington, D.C.: CCSDS, May 2012.

[8] M.J. Weinberger, G. Seroussi, and G. Sapiro. The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls. Image Processing, IEEE Transactions on, 9(8):1309–1324, 2000. ISSN 1057-7149. doi: 10.1109/83.855427.

[9] Xiaolin Wu and N. Memon. Context-based, adaptive, lossless image coding. Communications, IEEE Transactions on, 45(4):437–444, 1997. ISSN 0090-6778. doi: 10.1109/26.585919.

[10] S. Hunt and L.S. Rodriguez. Fast piecewise linear predictors for lossless compression of hyperspectral imagery. In Geoscience and Remote Sensing Symposium, 2004. IGARSS '04. Proceedings. 2004 IEEE International, volume 1, pages –312, 2004. doi: 10.1109/IGARSS.2004. 1369023.

[11] Xiaolin Wu and N. Memon. Context-based lossless interband compression-extending calic. Image Processing, IEEE Transactions on, 9(6):994–1001, 2000. ISSN 1057-7149. doi: 10.1109/83.846242.

[12] Lossless data compression recommended standard CCSDS 121.0-B-2. The Consultative Committee for Space Data Systems, 2012.

[13] Image data compression recommended standard CCSDS 122.0-B-1. The Consultative Committee for Space Data Systems, 2005.

[14] L. Santos, L.Berrojo, J. Moreno, J. López, R. Sarmiento, HyLoC: a Low-Complexity FPGA Implementation of the CCSDS-123 Standard Algorithm for Multispectral and Hyperspectral Compression, ESA-CNES On-Board Payload Data Compression Workshop, Venice, 2014

[15] García, A., Santos, L., López, S., Callicó, G. M., López, J. F., & Sarmiento, R. (2014, November). FPGA implementation of the hyperspectral Lossy Compression for Exomars (LCE) algorithm. In *SPIE Remote Sensing* (pp. 924705-924705). International Society for Optics and Photonics.