Aceleración HW con FPGA de la codificación de canal mediante el interfaz PCIe y DMA

Luis Rodríguez Cortés, José Tomás Entrambasaguas, Francisco J. Martín-Vega, Mari Carmen Aguayo-Torres
{lrodriguez, jtem, fjmvega, aguayo}@ic.uma.es.
Dpto. de Ingeniería de Comunicaciones. Universidad de Málaga, Málaga 29010, España.

***Resumen*—Hardware acceleration has gained renewed interest recently due to its great potential to solve the computational burdens associated with SW based implementation of real time communication systems. This approach considers SW based implementation of most of the communication functions due to its fast development cycles and great flexibility compared to HW development. However, those functions that are computationally demanding are implemented in HW to greatly reduce the computational time. Due to its iterative nature, channel encoding and decoding is related to high computational costs. This motivated us to develop a solution to this problem by offloading the encoding function to an FPGA. To minimize communication time between the computer and the FPGA, the PCIe interface is used, which achieves high transfer speeds. The results show the encoding of the message both in the computer and in the FPGA, along with their corresponding validation and performance metrics.**

I. Introducción

En el ámbito de las comunicaciones móviles, se busca aumentar la velocidad de los sistemas con el objetivo de reducir la latencia y transmitir la máxima información en el mínimo tiempo posible. En concreto, en los estándares recientes de telecomunicaciones (4G, 5G) hay ciertas tareas de la capa física como la codificación de canal iterativa o la detección multi-antena, que tienen un gran coste computacional. En este contexto, los aceleradores *hardware* (HW) han tomado cierta importancia recientemente, ya que permiten liberar carga de la parte *software* (SW) en la que se virtualizan las funciones de red, y realizan ciertas de estas funciones mucho más rápido [1]. Sin embargo, la implementación HW requiere mayor tiempo de desarrollo que la implementación SW, y además es menos flexible a cambios en el diseño. Es por este motivo que en los últimos años se ha optado por combinar ambos recursos, implementando una parte de la funcionalidad en SW, y la otra, más compleja, en HW.

En el contexto de *open radio access network* (O-RAN), el rendimiento es especialmente importante, debido a los ajustados requisitos de latencia y eficiencia energética entre otros. En concreto, las funciones de capa física se ven altamente beneficiadas por el uso de aceleradores HW, ya que requieren de una mayor capacidad computacional [2]. La combinación de O-RAN y 5G permite a las operadoras mejorar sus redes inalámbricas sin la necesidad de usar equipos específicos, ya que las funciones de red van virtualizadas sobre equipos de propósito general. Esto permite abrir el mercado a muchos más proveedores. Las soluciones basadas en O-RAN se dividen en tres unidades, centralizada, distribuida y de radio. En la unidad distribuida y la de radio, la aceleración HW permite un procesamiento más rápido de la capa física (L1) [3]. Dentro de O-RAN, en relación a la aceleración HW hay dos posibles arquitecturas, *look-aside* e *in-line*. La primera requiere de una *central processing unit* (CPU) conectada vía *Peripheral Component Interconnect Express* (PCIe) a una *Field Programmable Gate Array* (FPGA), que recibe los datos a procesar cuando sea necesario, mientras que en la segunda todos los datos pasan por el acelerador, que está en la misma placa que la CPU. En este proyecto haremos uso de la arquitectura *look-aside* [4] (enlace) [5] (enlace). Podemos ver un ejemplo de aceleración HW para O-RAN en Japón, donde se pretende lanzar una red 5G utilizando tarjetas gráficas de NVIDIA (*graphics processing units* (GPUs)) como aceleradores [6] (enlace).

El presente trabajo comprende el diseño y validación de un acelerador HW. En nuestro sistema, un ordenador envía a través del interfaz PCIe datos a codificar a una FPGA, que los devuelve codificados a través del mismo interfaz.

Otra posible aplicación de aceleradores HW es en el contexto de los gemelos digitales. Este campo aplicado a las comunicaciones requiere una latencia mínima, por lo que es ideal el uso de aceleradores con FPGAs o GPUs [7].

Hay que tener en cuenta que la aceleración HW pretende mejorar las prestaciones de un sistema, pero hay un tiempo de comunicación que no es despreciable. Si la latencia introducida por la comunicación mediante PCIe es mayor que el tiempo ganado por la aceleración, convendría no usar el acelerador HW.

Para este trabajo, el coprocesador considerado es un codificador de canal. La codificación de canal consiste en detectar y corregir errores de bits en sistemas de comunicaciones digitales. La codificación de canal se realiza tanto en el transmisor (codificador) como en el receptor (decodificador), y añade redundancia, por lo que es necesario un mayor ancho de banda para la transmisión de los datos. A cambio, muchos de los errores provocados por ruido o interferencias se pueden corregir en el receptor [8].

II. Aceleradores HW basados en PCIe

Un acelerador HW basado en PCIe es un dispositivo que se comunica con un ordenador mediante PCIe, para realizar tareas específicas que liberan la carga de la CPU principal y se ejecutan más rápido. Algunos ejemplos de esto serían GPUs, FPGAs o *Application-Specific Integrated Circuits* (ASICs), siendo el uso de PCIe clave para conseguir altas velocidades de transferencia de datos gracias a su gran ancho de banda.

La arquitectura de un sistema con acelerador HW considera que la CPU deriva algunos procesos como cálculos complejos o procesamiento de datos al dispositivo para acelerar su ejecución.

En nuestro caso, utilizaremos una FPGA como acelerador HW. La FPGA se conecta al ordenador como un periférico

a través de PCIe, y por tanto tiene asignado un puerto, una dirección y algunas señales de control. En nuestro caso, la FPGA está conectada mediante mapeado de memoria, por lo que tenemos direcciones separadas y líneas comunes para el plano de control. Los periféricos tienen tres maneras de comunicarse con el ordenador, mediante polling, interrupciones o DMA. Nosotros usaremos DMA, que implica transacciones de bloques de memoria sin la intervención de la CPU. El acceso al periférico es mediante el sistema operativo, por lo que en la llamada se produce una interrupción con su consecuente bloqueo. Una vez terminado el proceso, se desbloquea el proceso que estaba corriendo originalmente. Cada periférico tiene su propio controlador DMA, que es programable. Pero la programación de este controlador provoca una interrupción, aunque después de eso la transferencia se complete sin hacer uso de la CPU. Por este motivo, han surgido métodos como el *Data Plane Development Kit* (DPDK), que se salta el kernel de linux para procesar las transferencias de paquetes mucho más rápido [9].

III. Arquitectura propuesta

El sistema diseñado está compuesto por dos elementos principales, el ordenador y la FPGA, como podemos observar en la figura 3. En el ordenador tenemos por una parte el driver que gestiona la comunicación PCIe con DMA, y por otra, el SW de validación. En la FPGA tenemos un bloque que gestiona la comunicación mediante PCIe y DMA, y otro que se encarga del proceso de codificación, conectados entre sí mediante el interfaz AXI.

Este trabajo parte de un ejemplo de Xilinx que proporciona una comunicación simple mediante PCIe entre el ordenador y una memoria presente en la FPGA. A partir de ahí, se ha desarrollado un acelerador HW añadiendo varios bloques, duplicando la memoria y distribuyendo el interfaz AXI para tener un buffer de entrada y otro de salida. Además se ha implementado un bus AXI adicional y una capa de adaptación para la entrada y salida de datos del turbo codificador, incorporado de este trabajo [10].

El elemento xilinx_dma_pcie_ep es el bloque superior en el diseño del acelerador HW en la FPGA. Contiene los bloques que gestionan la comunicación PCIe, los bloques que gestionan la comunicación con el coprocesador mediante AXI, las memorias de entrada y salida, y la lógica de control implementada con máquinas de estados finitos.

En la figura 1 se presenta el interfaz del bloque xilinx_dma_pcie_ep, que se encarga de codificar mensajes de N bits, gestionando la comunicación mediante PCIe y DMA con el ordenador.

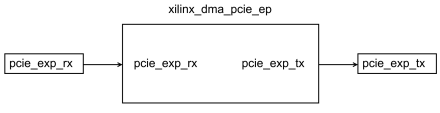

Fig. 1. Interfaz del bloque xilinx_dma_pcie_ep

En la figura 2 se presenta la estructura del bloque xilinx_dma_pcie_ep.

A continuación se describe el funcionamiento del bloque xilinx_dma_pcie_ep. El elemento está a la espera de bits a la

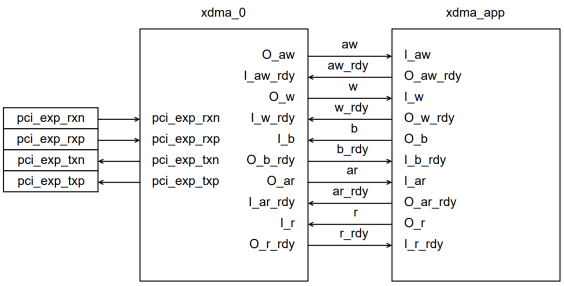

Fig. 2. Estructura del elemento xilinx_dma_pcie_ep

entrada, que entran al subbloque *xdma_0* encargado de gestionar la comunicación con el ordenador mediante PCIe. *xdma_0* se encarga de convertir la información recibida mediante PCIe a formato AXI, con el que se comunica con el subbloque *xdma_app*. Una vez *xdma_app* recibe un bloque completo, procede a codificar los datos y devolverlos codificados al bloque *xdma_0*, también mediante AXI. Por último, cuando *xdma_0* recibe una petición de lectura del ordenador por PCIe, convierte los datos AXI al formato PCIe y los envía de vuelta al ordenador.

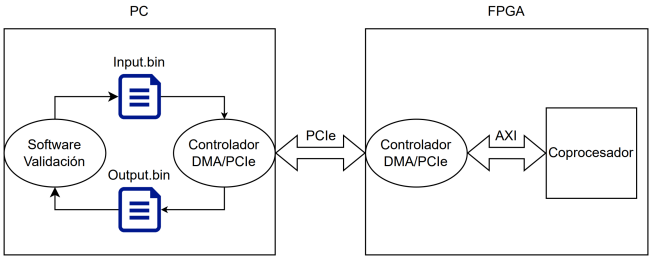

Fig. 3. Arquitectura del Sistema

El bloque *xdma_app* es un coprocesador encargado de recibir los datos por el interfaz AXI de entrada, codificarlos y devolverlos por el interfaz AXI de salida. La estructura del bloque se presenta en la figura 4. A continuación se expone el funcionamiento interno del bloque. El elemento está a la espera de una petición de escritura, que entrará al bloque *Interfaz_AXI_Entrada* a través del canal *aw*. Una vez recibida la petición, se rellena la memoria contenida en *Interfaz_AXI_Entrada* con los datos a codificar y se extraen la longitud del bloque y los parámetros. Una vez se ha rellenado la memoria de entrada, el bloque *Interfaz_AXI_Entrada* pone a '1' la señal *procesa_paquete*, indicando al bloque *Codif_Entrada* que ya puede comenzar la lectura de los datos. A continuación, el bloque *Codif_Entrada* lee los datos a través del canal *r*, y se los entrega al bloque *Codificador* en el formato adecuado (bit a bit). Además, una vez ha terminado de leer los datos, pone la señal *paquete_leido* a '1' para avisar a *Interfaz_AXI_Entrada* de que ya se puede recibir un nuevo bloque. El bloque *Codificador* se encarga de codificar los datos una vez ha entrado el bloque completo, y entrega los bits sistemáticos y de paridad de 3 en 3 (1 bit sistemático y los 2 correspondientes de paridad) al bloque *Codif_Salida*. El bloque *Codif_Salida* recibe los bits y los agrupa, para después enviarlos mediante el canal *w* al bloque *Interfaz_AXI_Salida*, y almacenarlos así en la memoria de salida. Además, pone a '1' la señal *paquete_procesado* una vez se terminan de escribir

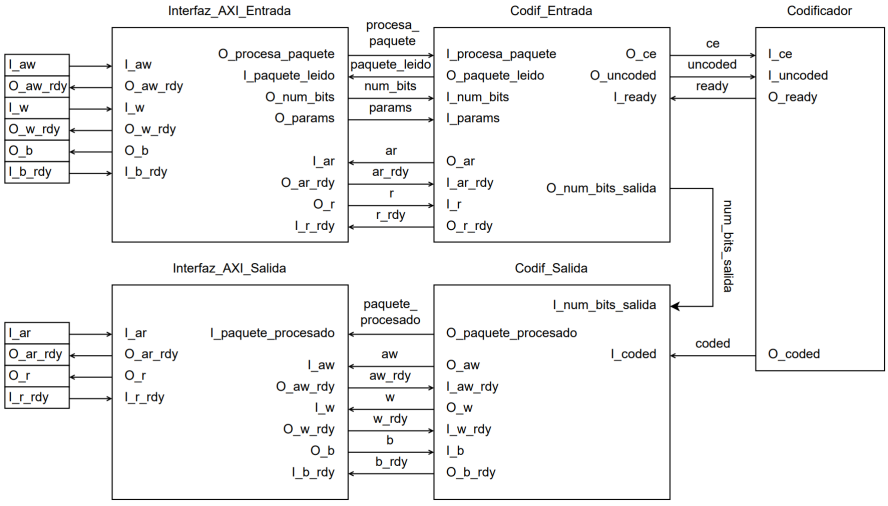

Fig. 4. Estructura del elemento xdma_app

los datos, para permitir su posterior lectura. Por último, el bloque *Interfaz_AXI_Salida* recibe una petición de lectura por el canal *ar*, y si ya ha recibido *paquete_procesado*, transmite los datos de la memoria de salida por el canal *r*.

IV. Resultados numéricos

Para probar las prestaciones del sistema se han hecho pruebas de rendimiento, consistentes en enviar varios bloques a la FPGA para codificarlos. Se han medido los tiempos haciendo uso de unas funciones escritas en C. Se han medido los tiempos para 1, 10, y 100 bloques de 40 bits (el tamaño más pequeño de bloque reflejado en el estándar de LTE [11]), y los mismos bloques de 6144 bits (el tamaño más grande de bloque reflejado en el estándar). Estas pruebas se han repetido 10000 veces cada una, obteniendo la distribución de probabilidad de los tiempos de codificación. Una vez realizadas las pruebas, se comparan los tiempos con los obtenidos utilizando la librería Sionna de Python para la codificación.

El objetivo de estas pruebas es caracterizar el comportamiento del sistema desarrollado y cuantificar los beneficios que pueda aportar como co-procesador. Las métricas que se han evaluado son estadísticos del retardo y del régimen binario. El gran beneficio que se quiere verificar consiste en reducir la carga computacional del SW que ejecuta las funciones de capa física y capas altas de un sistema de comunicaciones, y evaluar por tanto el aumento de régimen binario y reducción del retardo con respecto a una alternativa que implemente la decodificación únicamente en SW. Estos beneficios se espera que sean aún mayores en el caso de la decodificación, ya que esta tarea es más costosa computacionalmente.

En la figura 5 podemos observar la función de densidad de probabilidad del retardo en microsegundos para 1, 10 y 100 bloques codificados de 40 bits. El retardo mide el tiempo desde que los datos se envían a la FPGA hasta que vuelven codificados. La tónica general es que los valores de retardo se estabilizan más (y son más bajos) conforme aumentamos el número de paquetes enviados seguidos a la FPGA. En las tablas I y II observamos las diferencias de retardo y régimen binario entre la codificación SW usando Sionna y la codificación HW usando nuestro acelerador. Claramente existe una mejora al usar el acelerador, a pesar de estar haciendo las medidas para los paquetes de menor tamaño posible. En la tabla I vemos también la variación del retardo (desviación estándar), y se ve claramente cómo las muestras tienen mucha menos dispersión conforme aumentamos el número de paquetes enviados. Además, en la figura 5 observamos dos picos para N = 100, uno en 36.5 us y otro en 75.5 us. Esto se debe a la aleatoriedad de tiempos de las llamadas al sistema, que parecen más probables en esos valores. La función utilizada para estimar la función de densidad de probabilidad es la función ksdensity de Matlab.

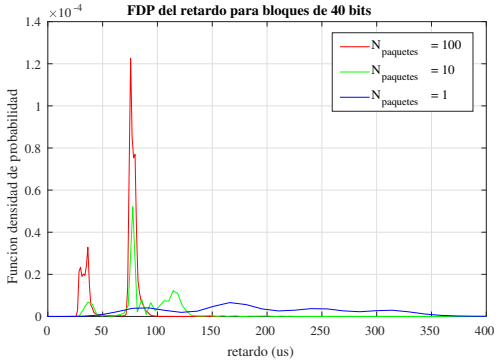

Fig. 5. Función Densidad de Probabilidad del retardo en bloques de 40 bits

N_paquetes	*Retardo*	*Desviación estándar*	*Régimen Binario*
1	200.9 us	109.1 us	0.20 Mbps
10	83.46 us	23.16 us	0.48 Mbps
100	66.16 us	19.69 us	0.61 Mbps

Tabla I: Métricas para bloques de 40 bits usando la FPGA

En la figura 6 podemos observar la función de densidad de probabilidad del retardo en microsegundos para 1, 10 y 100 bloques codificados de 6144 bits. Comparando con la figura 5, se observa una mayor estabilidad para 10 y 100 paquetes,

N_paquetes	*Retardo*	*Régimen Binario*
1	54 ms	740.7 bps
10	49 ms	816.3 bps
100	48 ms	833.3 bps

TABLA II: Métricas para bloques de 40 bits usando Sionna

con los valores de retardo más concentrados en torno a 85 microsegundos. En este caso, los valores de los picos para N = 100 están más próximos que en el caso de 40 bits, probablemente debido a un mayor tiempo de procesado en la FPGA. En las tablas III y IV se observan mayores diferencias de retardo y régimen binario entre la codificación SW y la codificación HW, haciendo aún más evidente la ventaja de usar un acelerador HW. Esta mejora en el régimen binario para paquetes de 6144 bits respecto a los de 40 se debe, a que la ventaja que obtenemos usando PCIe se vuelve más notoria conforme enviamos paquetes más grandes, por lo que enviando paquetes aún más grandes (por ejemplo con varios bloques a codificar en un mismo paquete) podríamos mejorar incluso más el régimen binario. También observamos que la desviación estándar del retardo se reduce mucho en el caso de N = 100 paquetes.

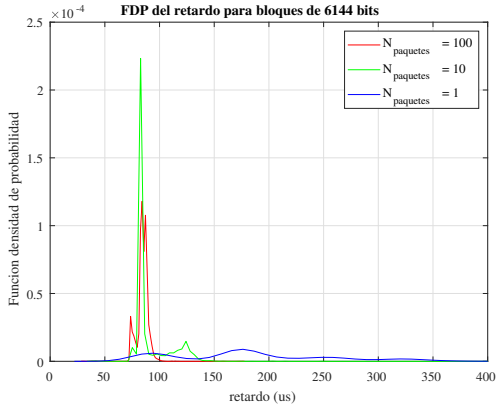

Fig. 6. Función Densidad de Probabilidad del retardo en bloques de 6144 bits

N_paquetes	*Retardo*	*Desviación estándar*	*Régimen Binario*
1	181.5 us	75.11 us	33.85 Mbps
10	93.1 us	18.71 us	65.99 Mbps
100	84.5 us	5.13 us	72.71 Mbps

TABLA III: Métricas para bloques de 6144 bits usando la FPGA

N_paquetes	*Retardo*	*Régimen Binario*
1	6.35 s	967.6 bps
10	6.30 s	975.2 bps
100	6.34 s	969.1 bps

TABLA IV: Métricas para bloques de 6144 bits usando Sionna

Comparando las tablas de 40 bits con las de 6144 bits podemos sacar las siguientes conclusiones. En primer lugar, queda claro que el acelerador HW mejora en varios órdenes de magnitud las prestaciones del codificador SW (sionna). En segundo lugar, comparando el régimen binario de 40 a 6144 bits en Sionna observamos que se mantiene más o menos constante, incluso empeora un poco al aumentar el tamaño de bloque. Esto se debe al coste de realizar el procesamiento por SW. Sin embargo, en el caso de la FPGA, la mayor parte del tiempo perdido es en la comunicación PCIe. Esto se hace patente si comparamos las tablas I y III, donde se observa que el aumento del tamaño de bloque apenas repercute en el retardo, consiguiendo un régimen binario mucho más alto con los bloques de 6144 bits. Por último, si comparamos la ganancia de la codificación HW frente a la SW, en el peor caso obtenemos una ganancia de 269, calculada dividiendo el retardo para un paquete de 40 bits en Sionna entre el retardo para 1 paquete de 40 bits en la FPGA. En el mejor caso, que sería dividiendo el retardo de 100 paquetes de 6144 bits en Sionna entre el retardo de 100 paquetes de 6144 bits en la FPGA, obtenemos una ganancia de 75030.

V. CONCLUSIONES

Se ha demostrado en este trabajo la reducción de tiempos de ejecución que otorga el enfoque de aceleración HW en la tarea de codificación de canal con respecto a una implementación puramente en SW. El enfoque planteado usa el interfaz PCIe para la comunicación, y a pesar de la latencia de la comunicación, se obtiene una reducción de los tiempos de cómputo. Así mismo, se ha observado que el retardo del procesado depende enormemente del número de paquetes a codificar, lo que sugiere que hay que diseñar de forma adecuada el método de envío de los bloques a la FPGA.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Fondo Europeo de Desarrollo Regional "FEDER Una manera de hacer Europa", MCIN/AEI/10.13039/501100011033 (España), Vodafone y la Universidad de Málaga a través de las subvenciones PID2022-137522OB-I00, RYC2021-034620-I y 8.06/6.10.6635.

REFERENCIAS

[1] M. Polese and et al., "Understanding o-ran: Architecture, interfaces, algorithms, security, and research challenges," *IEEE Communications Surveys Tutorials*, vol. 25, no. 2, 2023.

[2] L. Kundu, X. Lin, E. Agostini, V. Ditya, and T. Martin, "Hardware acceleration for open radio access networks: A contemporary overview," *IEEE Communications Magazine*, vol. 62, no. 9, pp. 160–167, 2024.

[3] S. Stanley, *Why open RAN needs flexible hardware acceleration*, 2021, https://www.lightreading.com/open-ran/why-open-ran-needs-flexible-hardware-acceleration.

[4] G. Owen, *Open RAN Networks – Layer 1 Acceleration Strategy Will Be Key To Operator Success*, 2023.

[5] M. A. Monem, *Why Physical Layer (L1) is a Big Deal in 5G Open RAN?*, 2022.

[6] M. N. CC Chong, Emeka Obiodu, *Enabling the World's First GPU-Accelerated 5G Open RAN for NTT DOCOMO with NVIDIA Aerial*, 2023.

[7] L. U. Khan, Z. Han, W. Saad, E. Hossain, M. Guizani, and C. S. Hong, "Digital twin of wireless systems: Overview, taxonomy, challenges, and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, pp. 2230–2254, 2022.

[8] S. Faruque, *Introduction to Channel Coding*. Cham: Springer International Publishing, 2016, pp. 1–16. [Online]. Available: https://doi.org/10.1007/978-3-319-21170-1_1

[9] T. L. Foundation, *Data Plane Development Kit (DPDK) documentation*, 2024, https://doc.dpdk.org/guides/index.html.

[10] F. Martín-Vega, F. J. López-Martínez, and J. T. Entrambasaguas, "Arquitectura HW para decodificador Two-Step SOVA con recorridos hacia atrás sistólicos, (documento acesible en: https://hdl.handle.net/10630/23749)," in *XXVII Simposium Nacional de la Unión Científica Internacional de Radio (URSI 2012)*, 2012, pp. 1–4.

[11] ETSI, *LTE; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (3GPP TS 36.212 version 18.0.0 Release 18)*, 2024.